LIMEIRA 2019

51º SBPO

Problema de Corte: modelagem matemática e resolução com auxílio de solver

Kelly Poldi

Carla Ghidini

UNICAMP

IMECC

FCA UNICAMP • LIMEIRA

Apresentação

- O CPLEX é um resolvedor de programação matemática de alto desempenho para:
  - programação linear;
  - programação inteira mista;
  - programação quadrática;
  - etc.

Instalação do CPLEX (versão acadêmica)

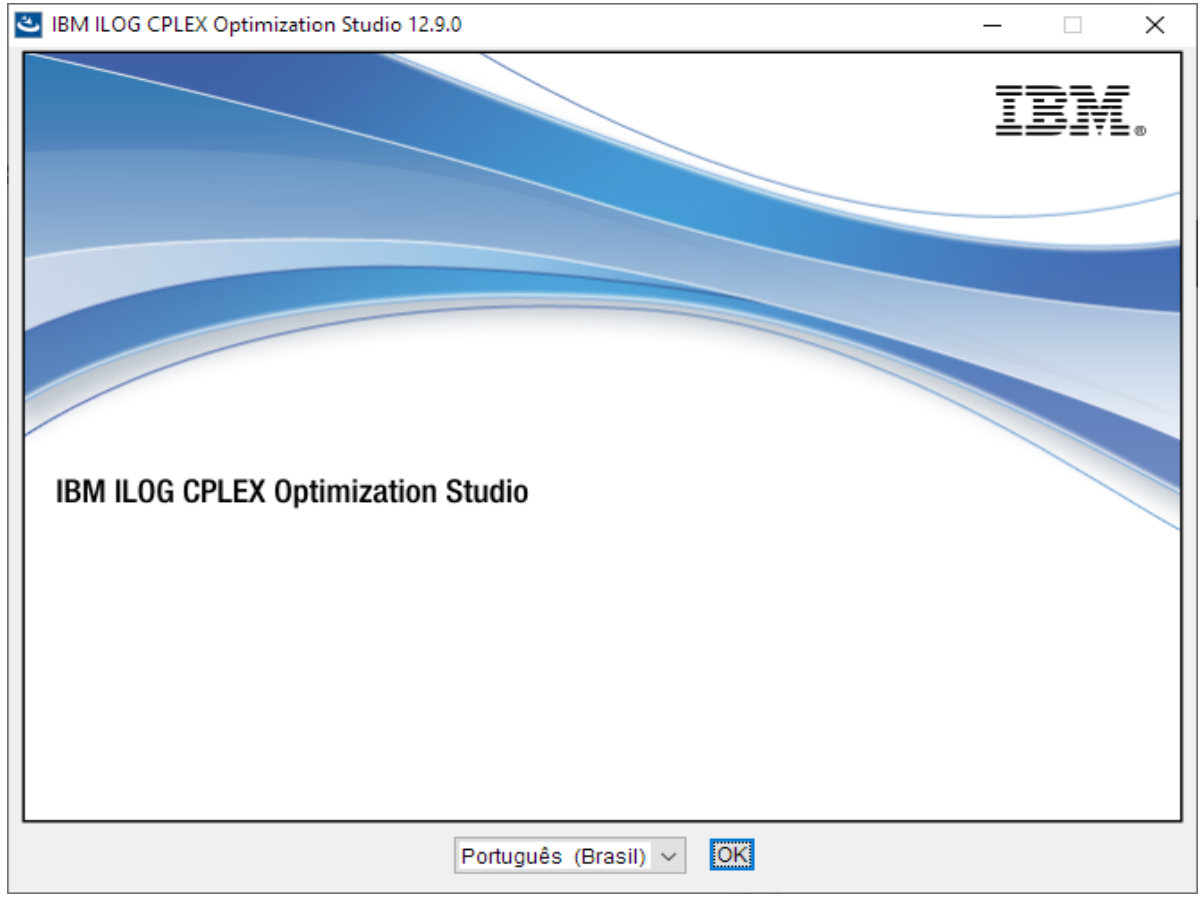
- Site: ibm.biz/academic.
- Se não possuir uma conta na IBM Academic clique no botão **Register now**.

  É preciso um e-mail de uma instituição de ensino reconhecida pela IBM.

  Caso contrário **Already registered? Log in.**
- Após o login, clique em **Data Science.**
- Dentro da caixa **ILOG CPLEX Optimization Studio**, clique em **Download v12.9.**
- Marque a caixa de seleção **IBM ILOG CPLEX Optimization Studio 12.9 for Windows x86-64 Multilingual (CNZM1ML).**
- No final da página, clique em **Download now** para iniciar o download.

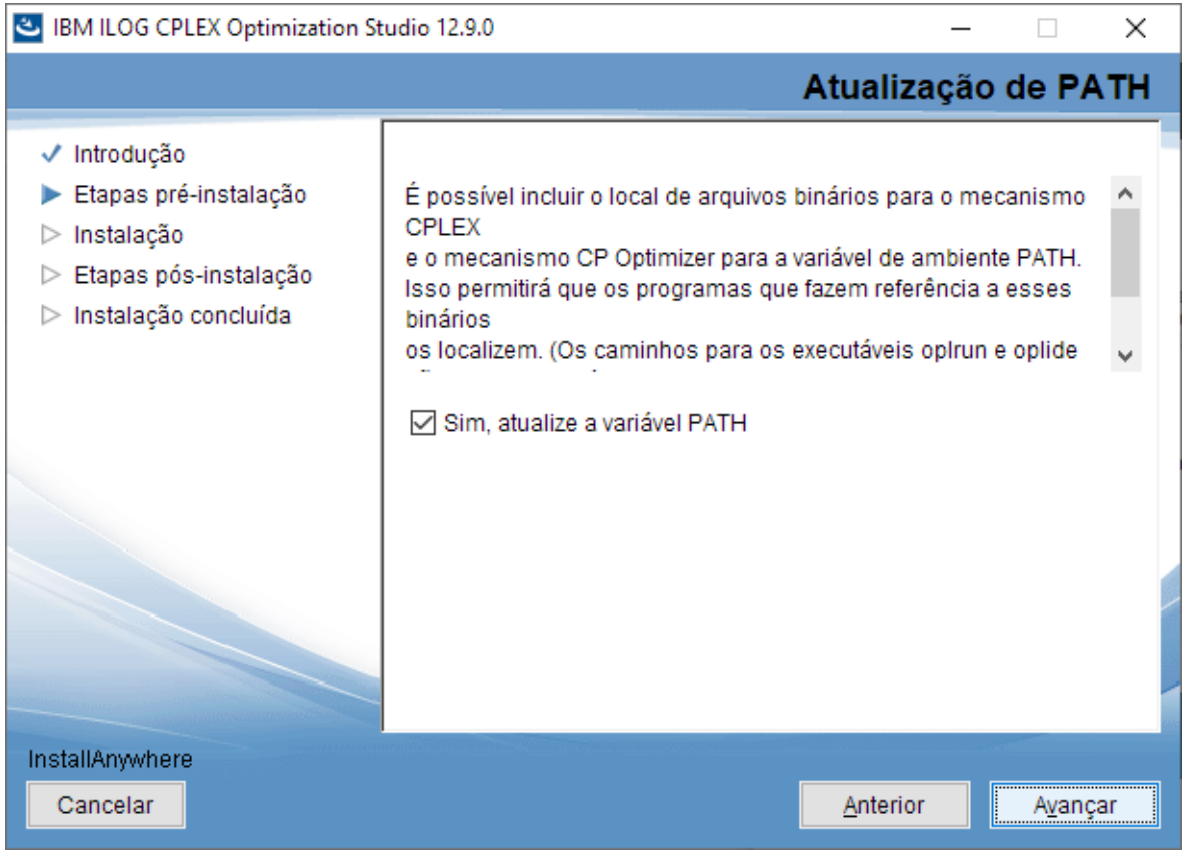
Instalação do CPLEX (versão acadêmica)

- Após o download ser concluído, execute o arquivo: **cplex_studio129.win-x86-64**



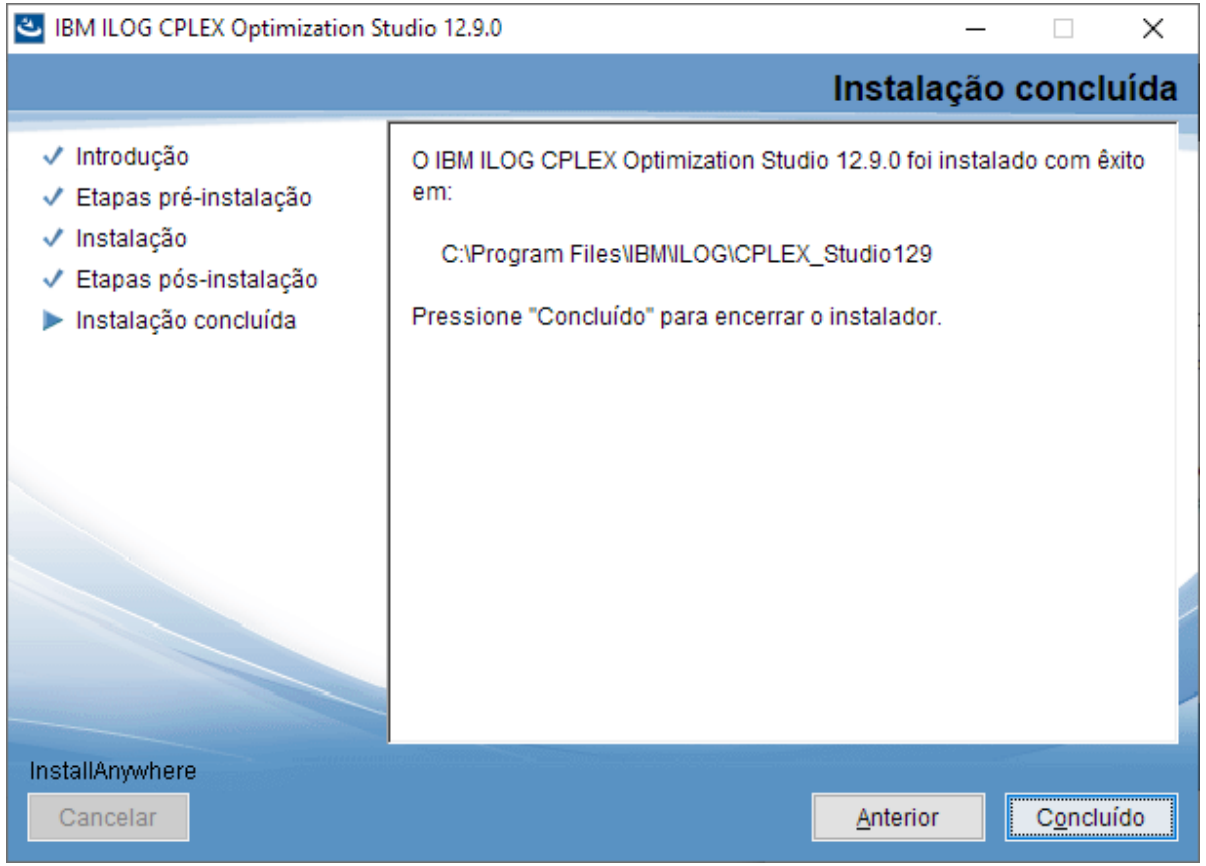
Instalação do CPLEX (versão acadêmica)

- Avance com a instalação.
- Na etapa **Atualização de PATH**, é interessante manter a caixa de seleção **Sim, atualize a variável PATH** marcada para que os programas que fazem referência ao CPLEX Callable Library o localize.



Instalação do CPLEX (versão acadêmica)

- Avance com a instalação até que ela seja concluída.



Formatos de arquivos suportados para programação linear

- MPS - *Mathematical Programming System*
  - Padrão de mercado.
  - Estabelecido há bastante tempo e tornou-se um padrão amplamente aceito para definir problemas de programação linear.
  - Formato orientado por coluna.

Formatos de arquivos suportados para programação linear

- MPS - *Mathematical Programming System*

```
NAME          example.mps
ROWS
 N  obj
 L  c1
 L  c2
COLUMNS
    x1        obj                -1   c1                -1
    x1        c2                  1
    x2        obj                -2   c1                 1
    x2        c2                 -3
    x3        obj                -3   c1                 1
    x3        c2                  1
RHS
    rhs       c1                 20   c2                30
BOUNDS
 UP BOUND     x1                 40
ENDATA
```

Formatos de arquivos suportados para programação linear

- MPS - *Mathematical Programming System*
  - Padrão de mercado.
  - Estabelecido há bastante tempo e tornou-se um padrão amplamente aceito para definir problemas de programação linear.
  - Formato orientado por coluna.

- LP - *CPLEX LP file format*
  - Padrão criado para uso com o resolvedor CPLEX.
  - Mais amigável que o formato MPS.
  - Compatível com os principais resolvedores modernos.

Formato LP

- **ESTRUTURA:**

  - 1. Função Objetivo
  - 2. Conjunto de Restrições
  - 3. Informações sobre as Variáveis de Decisão
    - Limites (bounds).
    - Tipos: inteiras, binárias.

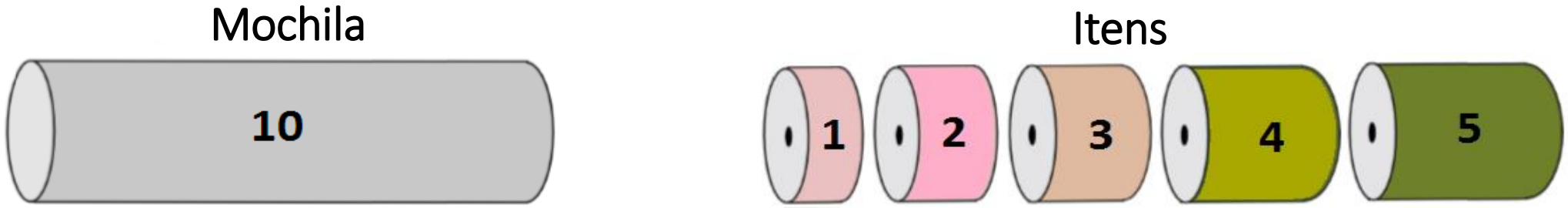
Formato LP

- **PROBLEMA 1:** Considere que temos disponíveis uma mochila com capacidade $L = 10$ e 5 tipos de itens ($m = 5$), cujos valores de utilidade e pesos são os seguintes:

  - $v = (4\ \ 6\ \ 5\ \ 3\ \ 1)^T$
  - $p = (5\ \ 4\ \ 3\ \ 2\ \ 1)^T$

```
Max   4x1 + 6x2 + 5x3 + 3x4 + x5

s.a.  5x1 + 4x2 + 5x3 + 2x4 + x5 ≤ 10
      x1, x2, x3, x4, x5 ≥ 0 e inteiras
```



Formato LP – Problema linear

```
\Problem name: Mochila.lp

Maximize
obj: 4x1 + 6x2 + 5x3 + 3x4 + x5

Subject To
c1: 5x1 + 4x2 + 3x3 + 2x4 + x5 <= 10

End
```

- OBSERVAÇÕES:
  - “Problem name:”, “obj:” e “c1:” não precisam ser escritos.
  - Ajuda na identificação das componentes do problema.

Formato LP – Problema linear inteiro

```
\Problem name: MochilaInteiro.lp

Maximize
obj: 4x1 + 6x2 + 5x3 + 3x4 + x5

Subject To
c1: 5x1 + 4x2 + 3x3 + 2x4 + x5 <= 10

Generals
x1 x2 x3 x4 x5

End
```

Formato LP – Problema linear binário

```
\Problem name: MochilaBinario.lp

Maximize
obj: 4x1 + 6x2 + 5x3 + 3x4 + x5

Subject To
c1: 5x1 + 4x2 + 3x3 + 2x4 + x5 <= 10

Binaries
x1 x2 x3 x4 x5

End
```

Formato LP – Variáveis canalizadas

```
\Problem name: MochilaCanalizada.lp

Maximize
obj: 4x1 + 6x2 + 5x3 + 3x4 + x5

Subject To
c1: 5x1 + 4x2 + 3x3 + 2x4 + x5 <= 10

Bounds
0 <= x1 <= 2
0 <= x5 <= 1

End
```

Ambiente interativo do CPLEX

Prompt

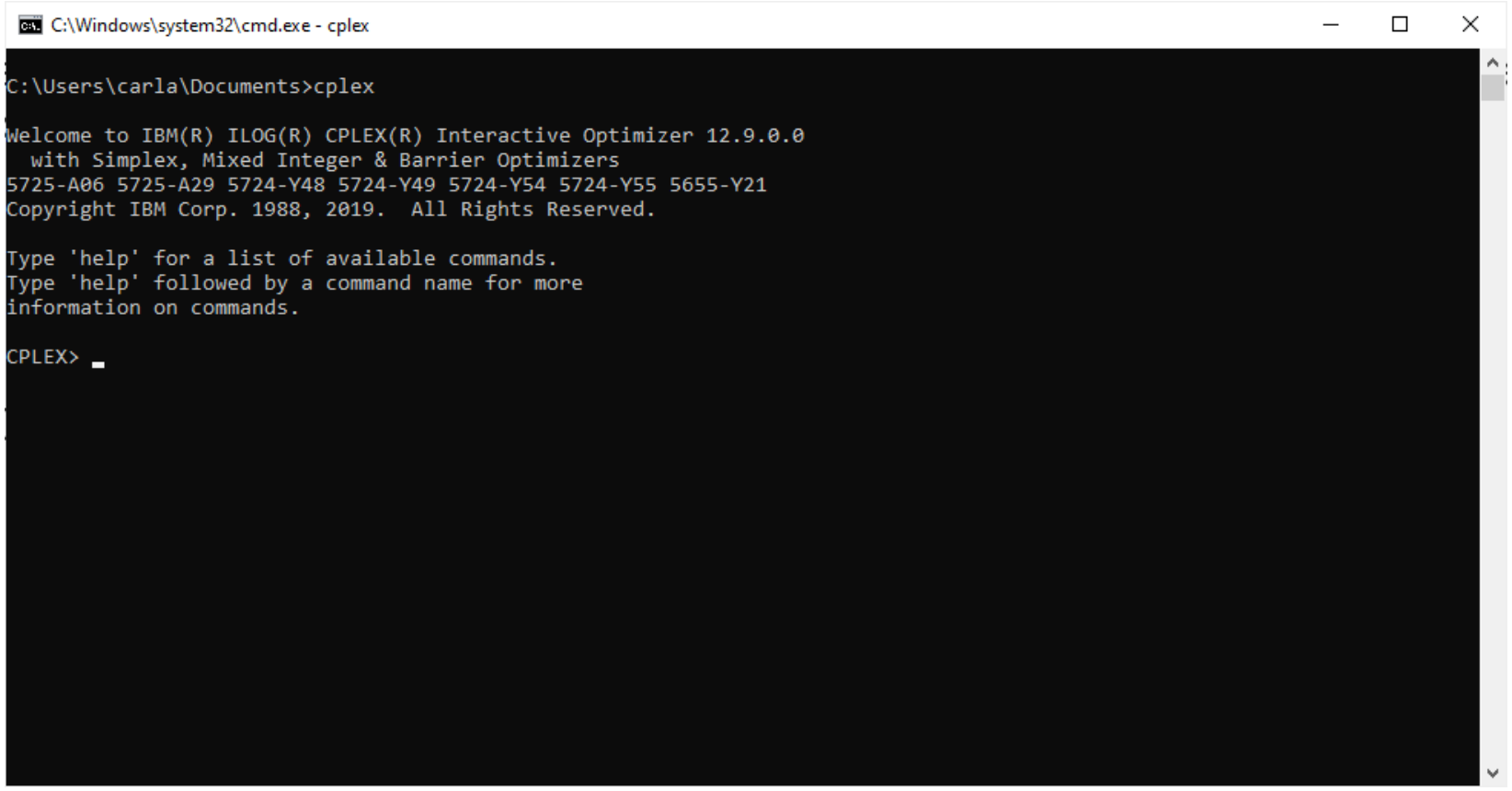
- Para iniciar o ambiente interativo:
  - Abra o ***Prompt* de Comando**
  - Digite **cplex** e pressione Enter para abrir o *prompt* do CPLEX.

    CPLEX>

Ambiente interativo do CPLEX

Prompt

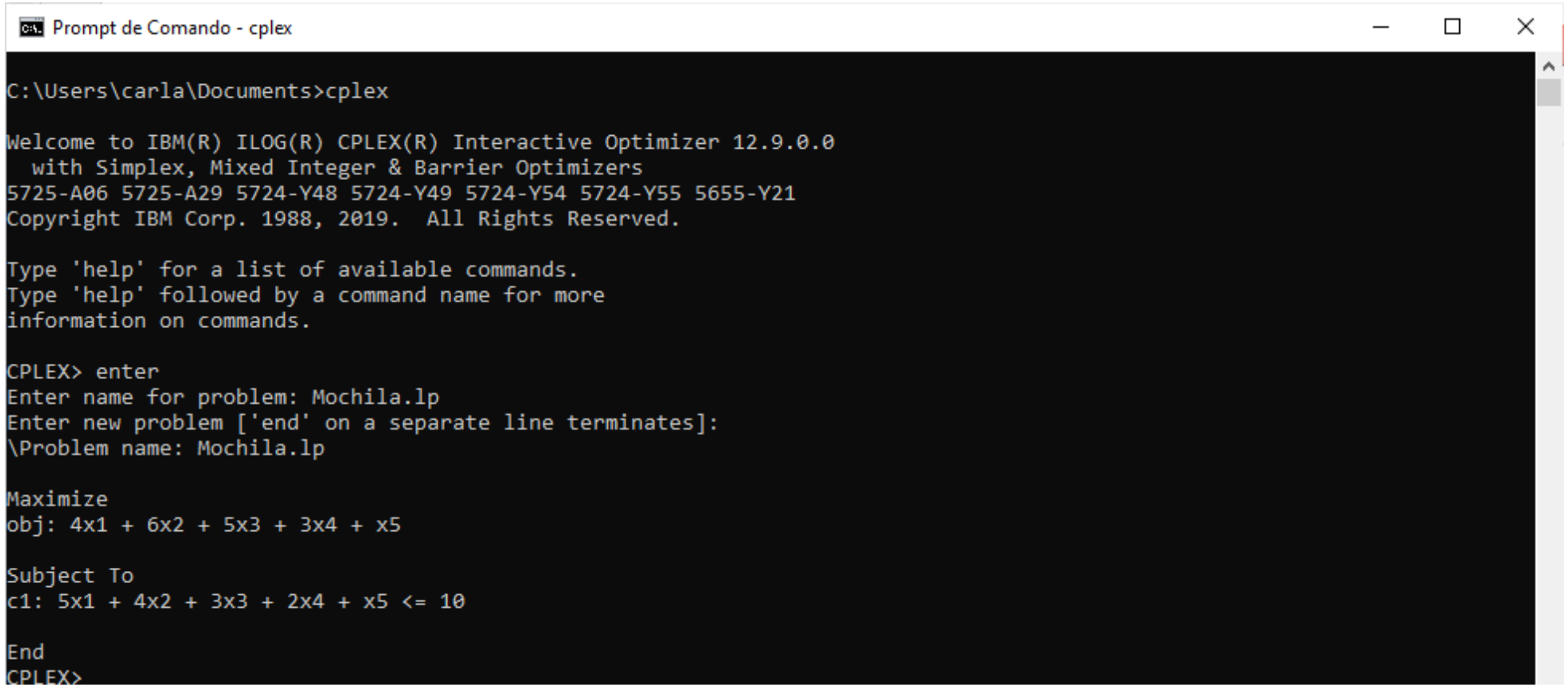
- Neste *prompt* é possível colocar diretamente um problema no formato LP e resolvê-lo posteriormente.



Ambiente interativo do CPLEX

Prompt

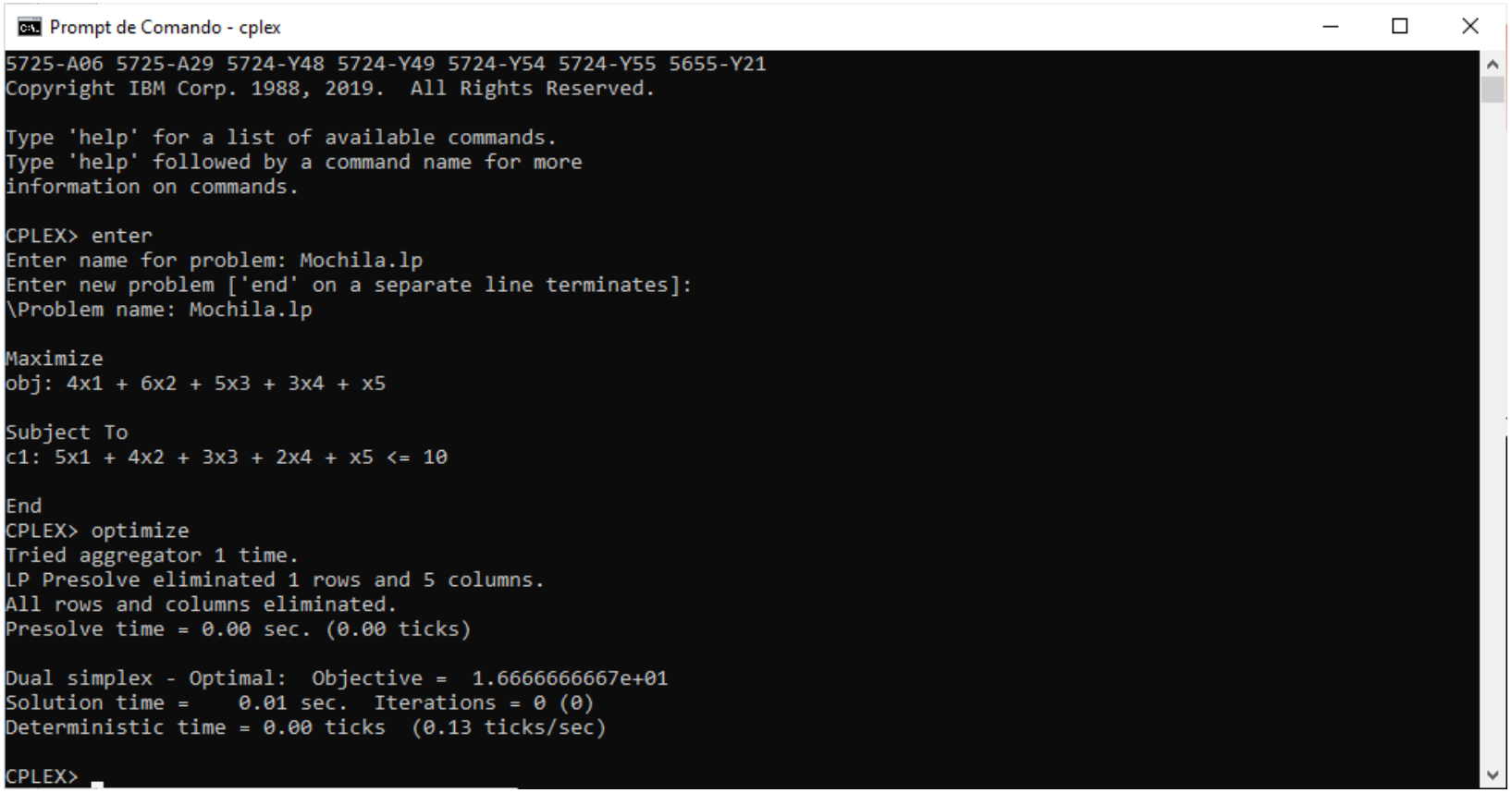
- O comando **enter nome_modelo** permite que o usuário digite todo o problema no formato LP, retornando ao *prompt* após o *End* do formato.



Ambiente interativo do CPLEX

Prompt

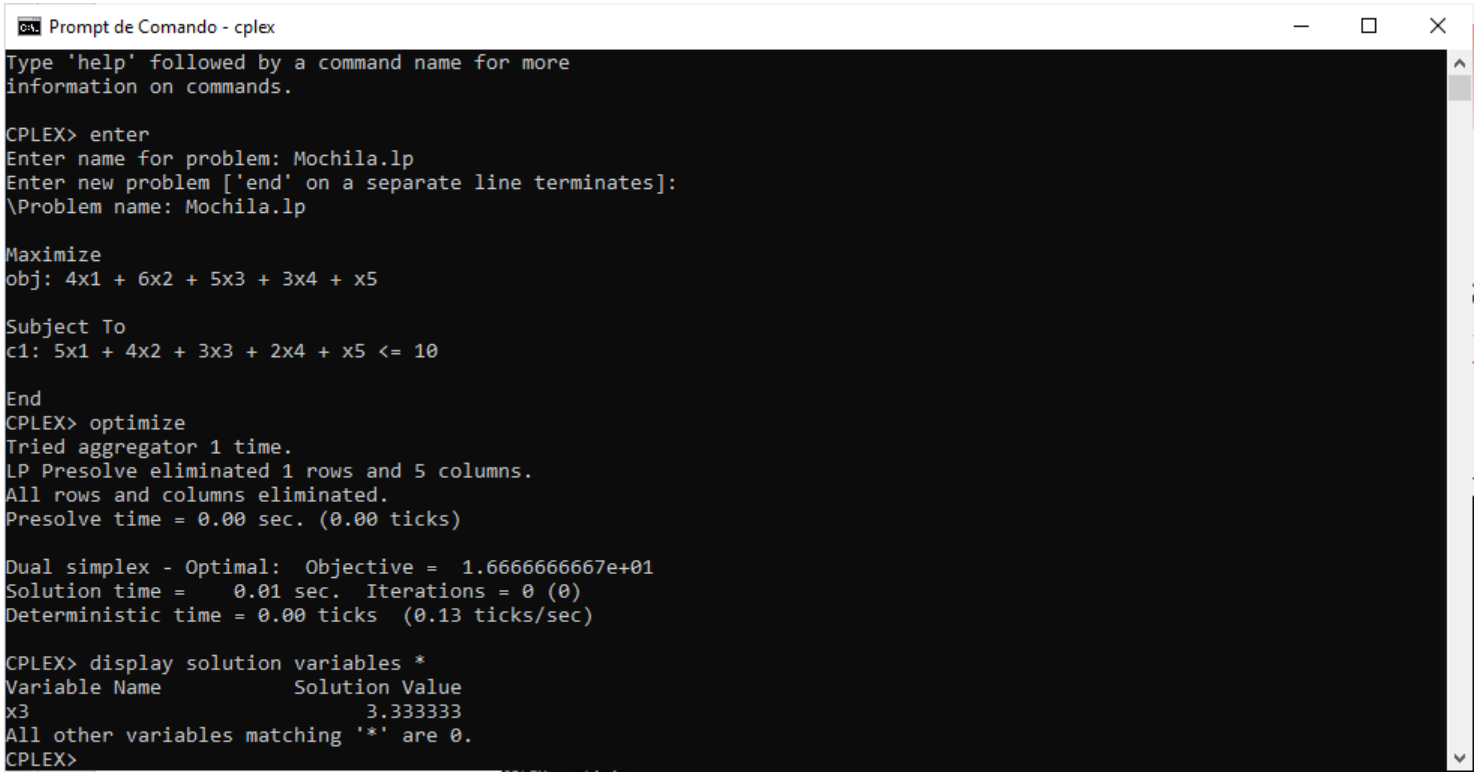
- Para resolver o modelo fornecido, utilize o comando **optimize.**



Ambiente interativo do CPLEX

Prompt

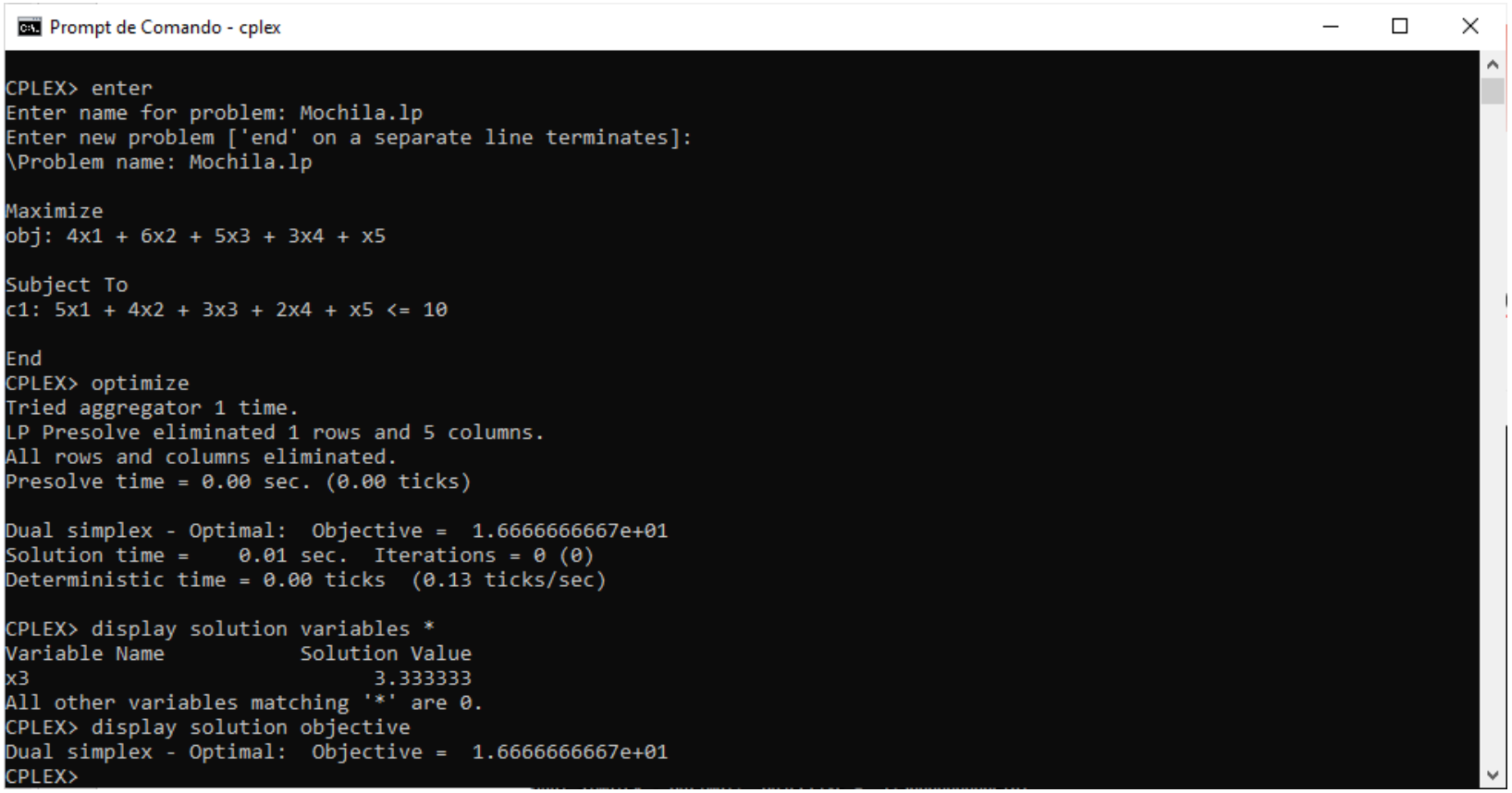
- Após a resolução do problema, pode-se verificar os valores das variáveis com o comando **display solution variables nome_variavel** (*nome_variavel* pode ser * para que sejam exibidos os valores de todas variáveis).



Ambiente interativo do CPLEX

Prompt

- Para exibir o valor da função objetivo, utilize o comando **display solution objective**



Ambiente interativo do CPLEX

Prompt

- OBSERVAÇÕES:

  - Escrever o problema diretamente no *prompt* não é muito recomendado, pois ele não será salvo.

  - É possível ler o problema salvo em um arquivo no formato LP. Para isso, utilize o comando **read nome_arquivo.lp**

Alguns comandos básicos

COMANDO	DESCRIÇÃO
add	add constraints to the problem
display	display problem, solution or parameter settings
enter	enter a new problem
help	provide information on CPLEX commands
mipopt	solve a mixed integer program
optimize	solve the problem
primopt	solve using the primal method
quit	leave CPLEX
read	read problem or basis information from a file
set	set parameters
tranopt	solve using the dual method
write	write problem or solution info. to a file

Apresentação

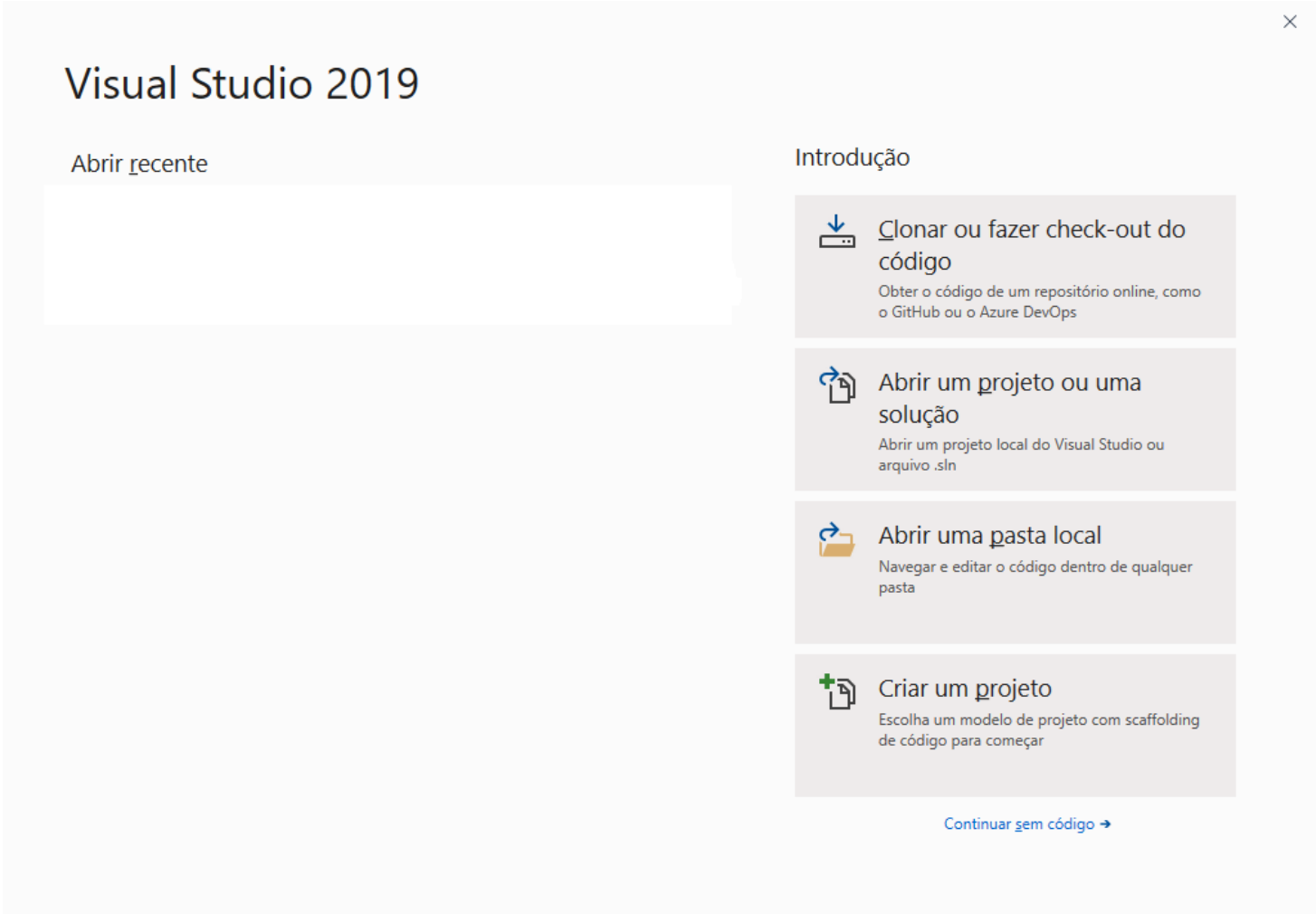
- **CPLEX Callable Library** é uma biblioteca em C que permite ao programador incorporar os otimizadores do CPLEX em aplicações escritas em C, Visual Basic, FORTRAN ou qualquer outra linguagem que pode chamar funções em C.

- Neste minicurso utilizaremos o Visual Studio 2019.

  - https://visualstudio.microsoft.com/vs/community/

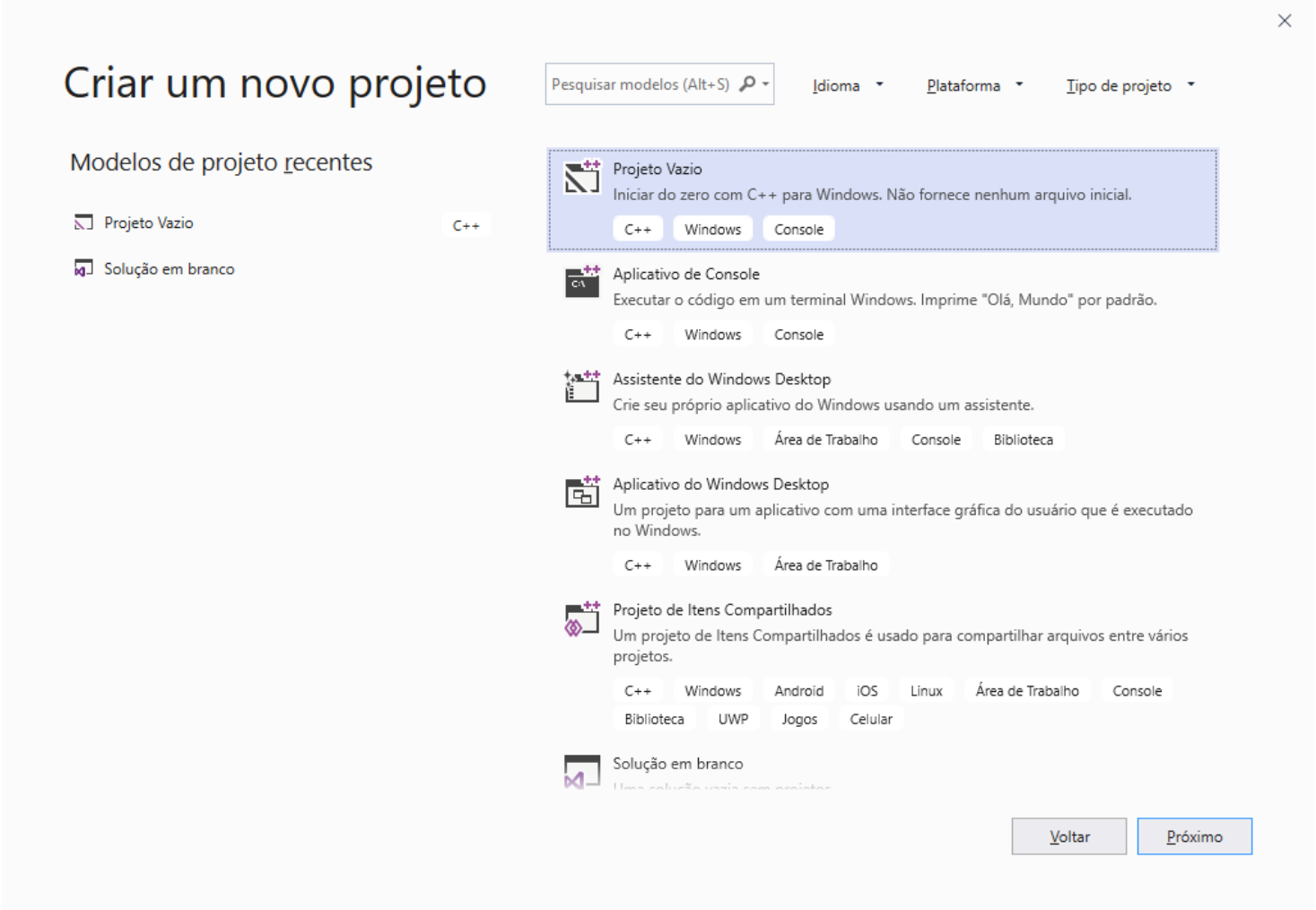
Criar projeto

- Clique em **Criar um projeto**.



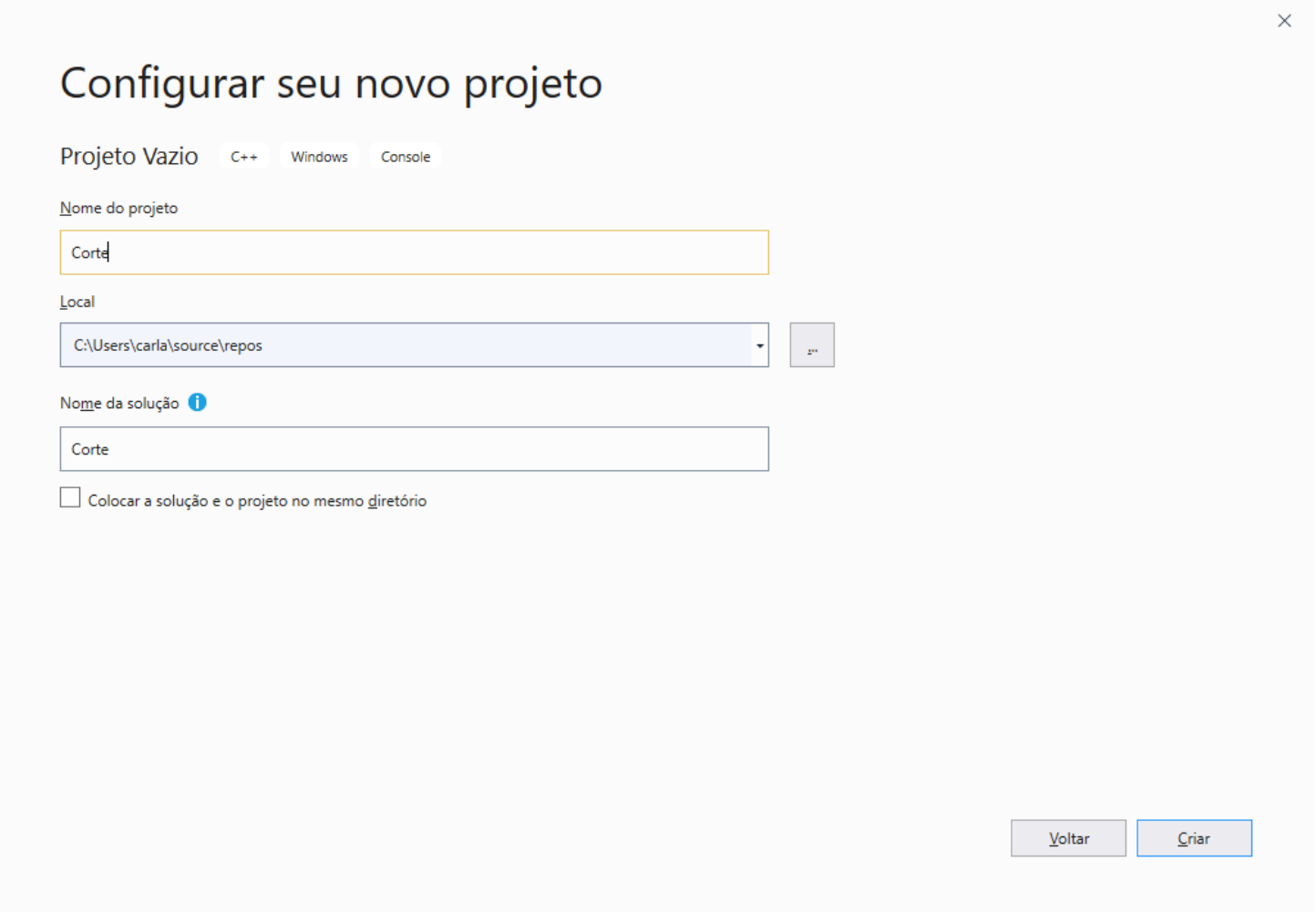
Criar projeto

- Clique em **Projeto vazio** e em **Próximo** para avançar.



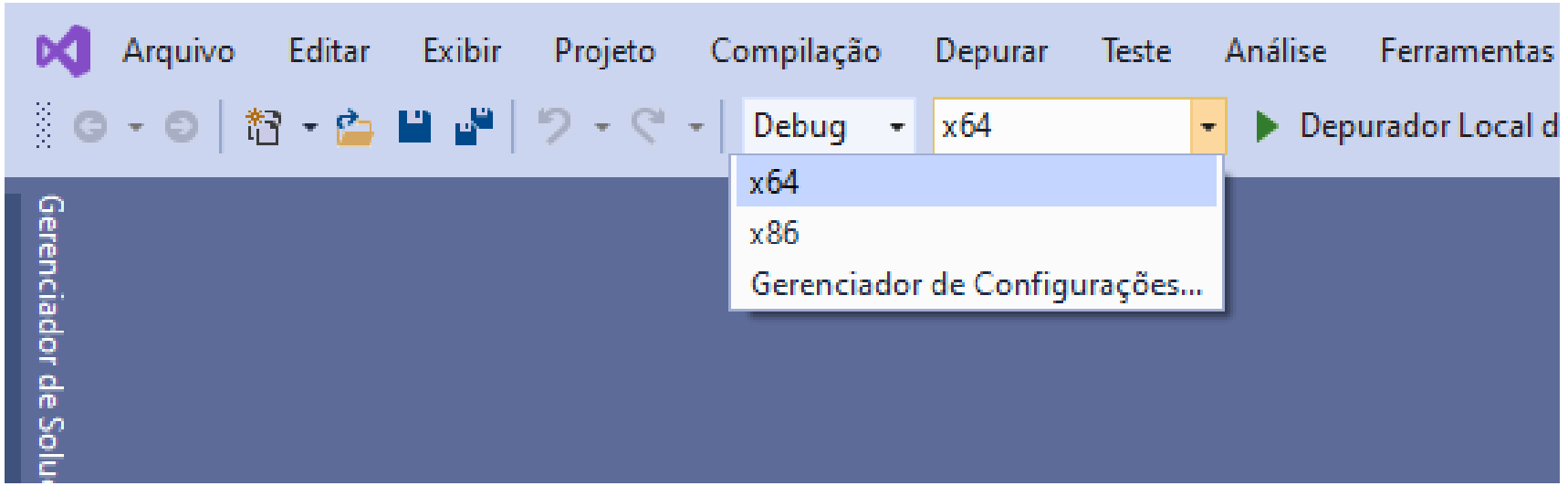
Criar projeto

- Em **Nome do projeto** digite Corte e em seguida clique em **Criar**.



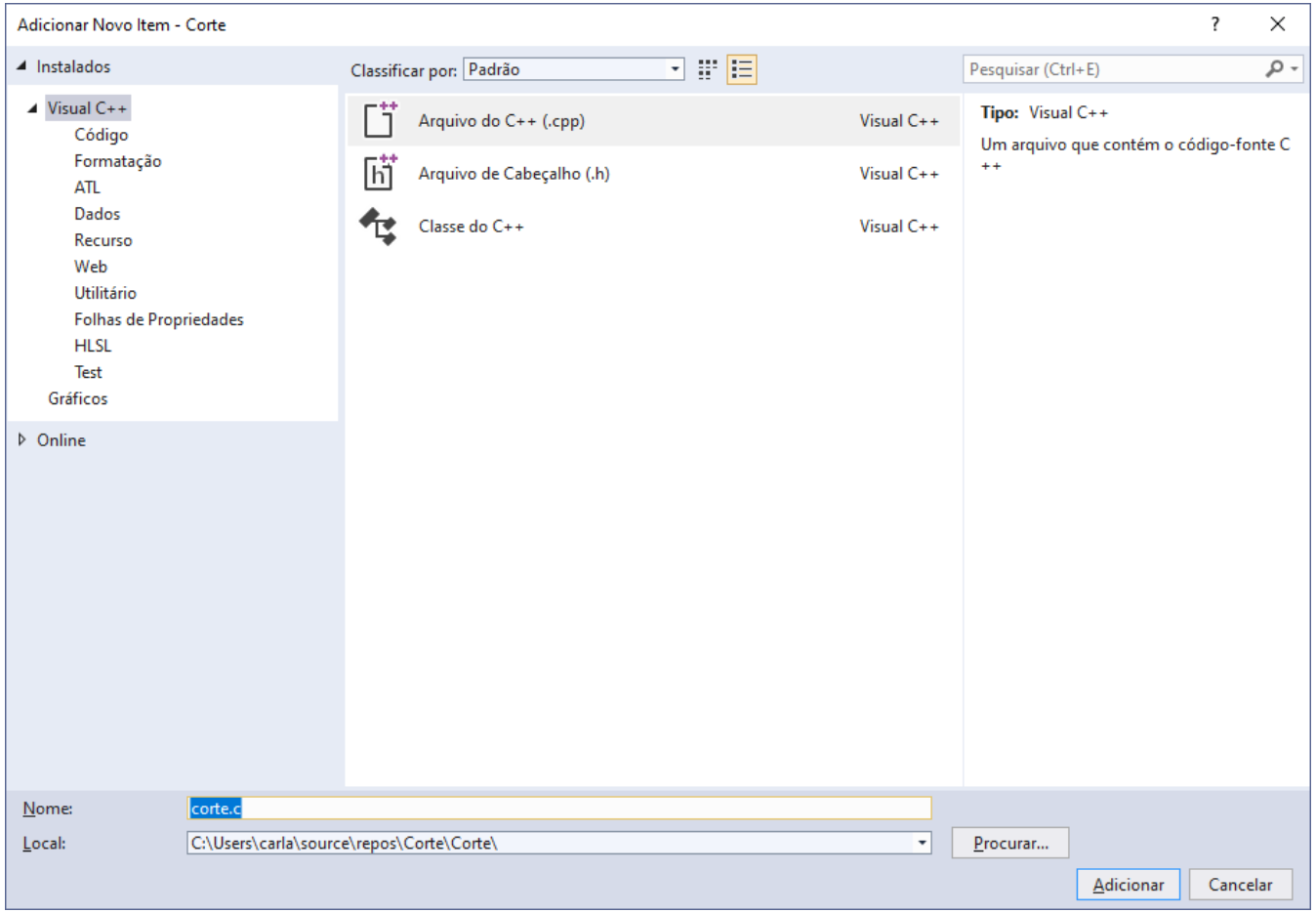
Criar projeto

- Na barra de ferramentas selecione **x64**.



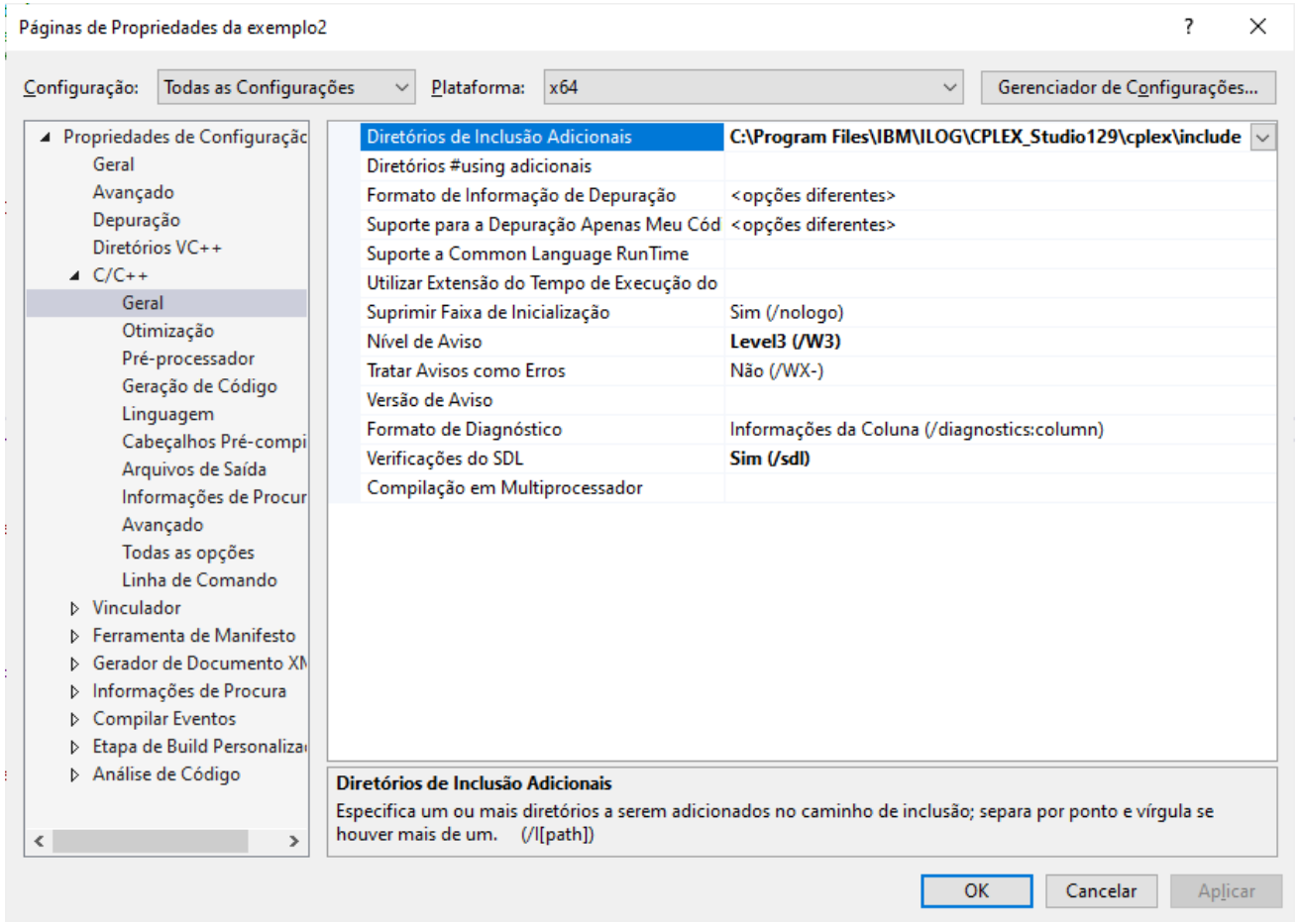
Criar projeto

- No menu **Projeto**, clique em **Adicionar Novo Item**. Digite *corte.c* em **Nome** e clique em **Adicionar**.



Criar projeto

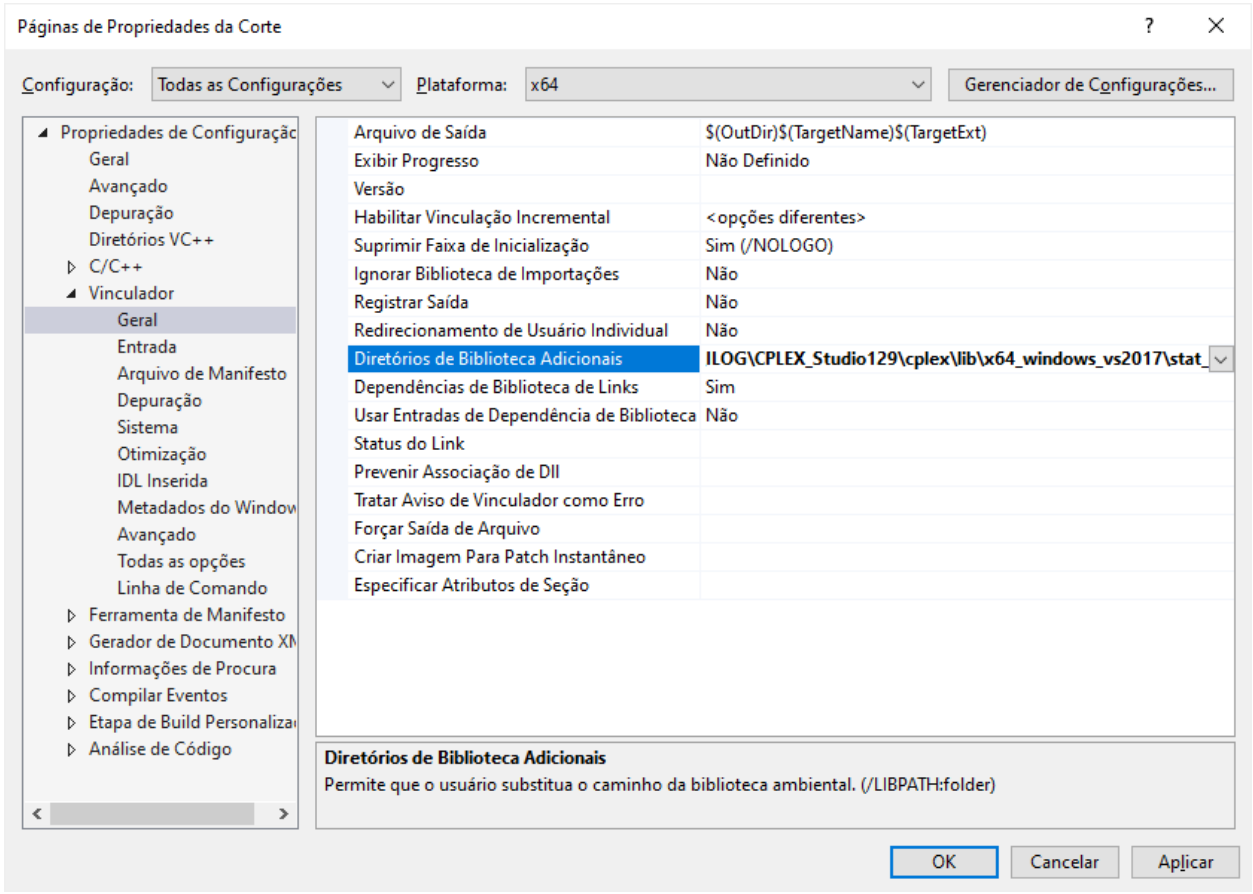
- No menu **Projeto**, clique em **Propriedades**. Selecione **C/C++**, **Geral** e em **Diretórios de Inclusão Adicionais** digite *C:\Program Files\IBM\ILOG\CPLEX_Studio129\cplex\include*



Criar projeto

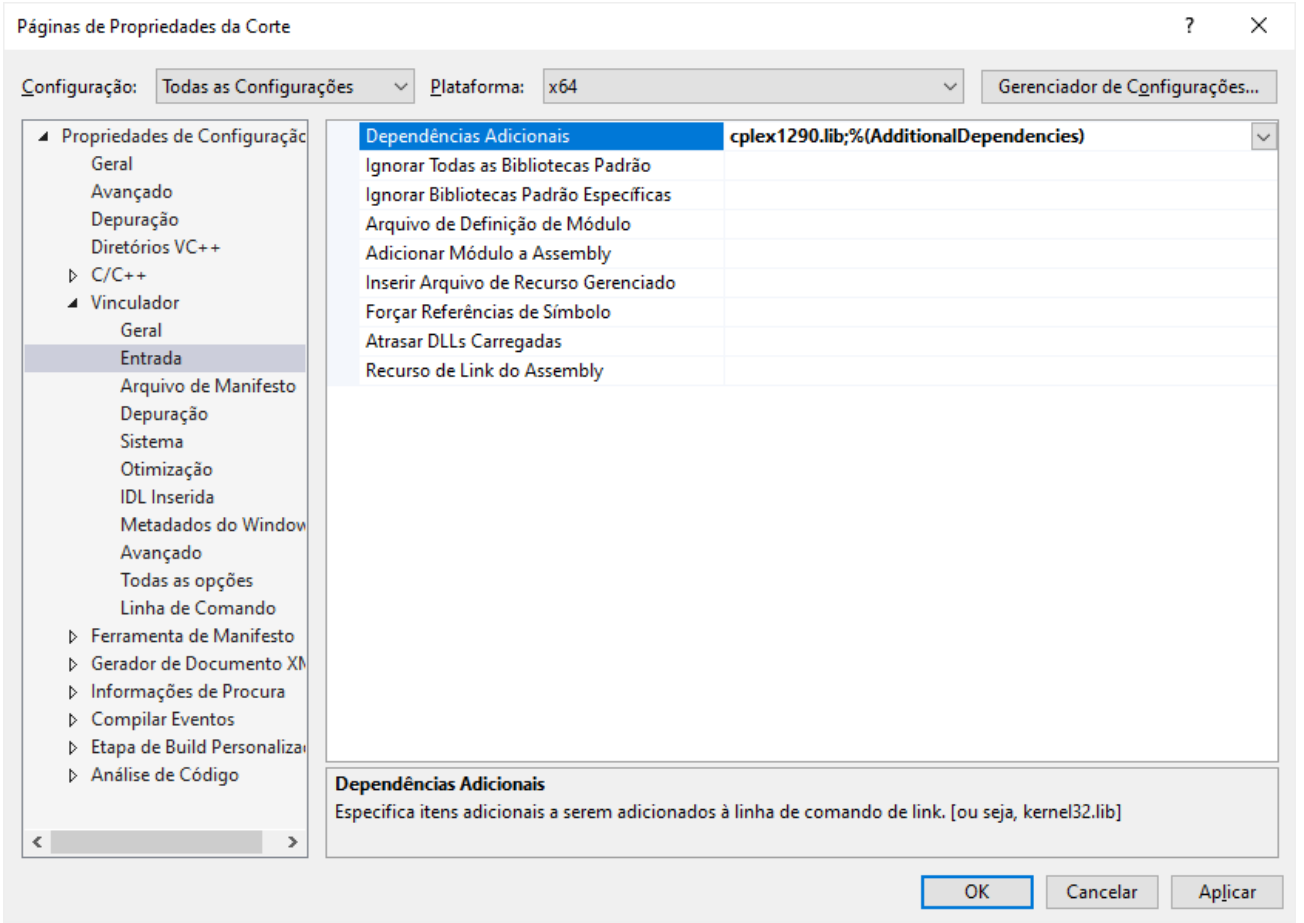
- Selecione **Vinculador**, **Geral** e em **Diretórios de Bibliotecas Adicionais** digite:

  C:\Program Files\IBM\ILOG\CPLEX_Studio129\cplex\lib\x64_windows_vs2017\stat_mdd



Criar projeto

- Selecione **Entrada** e em **Dependências Adicionais** clique em **Editar** e digite: *cplex1290.lib*
  Clique em **Ok** para finalizar.



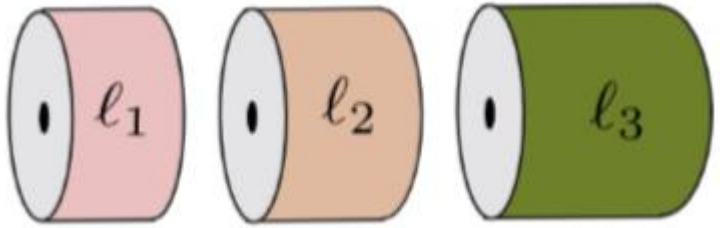
Problema de corte unidimensional

- **PROBLEMA 2:** Considere o problema de corte unidimensional com um único tipo de objeto em estoque de comprimento $L = 11$ a ser cortado para produzir três tipos de itens de comprimentos $\ell_1 = 3$cm, $\ell_2 = 4$cm e $\ell_3 = 5$cm e cujas demandas são 4, 3 e 2, respectivamente.

Objeto

11cm

Itens



3cm 4cm 5cm

Demanda

$d_1 = 4$
$d_2 = 3$
$d_3 = 2$

Problema de corte unidimensional

- **PROBLEMA 2:** Considere o problema de corte unidimensional com um único tipo de objeto em estoque de comprimento L = 11 a ser cortado para produzir três tipos de itens de comprimentos ℓ_1 = 3cm, ℓ_2 = 4cm e ℓ_3 = 5cm e cujas demandas são 4, 3 e 2, respectivamente.

Padrão de corte	1	2	3	4	5	6	7	8	9	10	11	12	13
Item 1	3	2	1	0	0	0	0	2	2	1	1	1	0
Item 2	0	0	0	2	1	0	0	1	0	2	1	0	1
Item 3	0	0	0	0	0	2	1	0	1	0	0	1	1
Perda	**2**	**5**	**8**	**3**	**7**	**1**	**6**	**1**	**0**	**0**	**4**	**3**	**2**

Problema de corte unidimensional

- **PROBLEMA 2:** Problema no Formato lp

```
\Problem name: Corte.lp

Minimize
obj: x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10 + x11 + x12 + x13

Subject To
c1: 3x1 + 2x2 + x3 + 2x8 + 2x9 + x10 + x11 + x12 >= 4
c2: 2x4 + x5 + x8 + 2x10 + x11 >= 3
c3: 2x6 + x7 + x9 + x12 + x13 >= 2

End
```

Inicializar o ambiente

- Para poder executar as funções da *Callable Library*, precisamos primeiro inicializar o ambiente do CPLEX (chamado de *env*) utilizando a função **CPXopenCPLEX**.

- Esta função retorna um ponteiro para o ambiente do CPLEX.

```
CPXENVptr env = NULL;
env = CPXopenCPLEX(&status);
```

Visualizar as saídas

- Para visualizar as saídas do CPLEX é preciso ligar o parâmetro **CPX_PARAM_SCRIND** utilizando a função **CPXsetintparam**.

```
status = CPXsetintparam(env, CPXPARAM_ScreenOutput, CPX_ON);
```

Criar o objeto problema

- Antes de ler o problema a ser resolvido, é preciso criar o objeto *problema* que irá representá-lo no CPLEX.

- A função **CPXcreateprob** cria, inicializa e retorna um ponteiro para um novo objeto *problema* vazio.

```
CPXLPptr lp = NULL;
lp = CPXcreateprob(env, &status, argv[1]);
```

Ler um arquivo no formato LP

- A função **CPXreadcopyprob** lê um arquivo no formato LP ou MPS e copia os dados para o objeto *problema*.

```
status = CPXreadcopyprob(env, lp, argv[1], NULL);
```

Otimizador

- Após o objeto problema ter sido definido, a função **CPXlpopt** pode ser usada para resolvê-lo. O *default* dessa função é o método *dual simplex*.

```
status = CPXlpopt(env, lp);
```

Alocar memória

- Antes de obter a solução é preciso alocar memória para receber os valores das variáveis primais, duais, folga e custo relativo.

```
cur_numrows = CPXgetnumrows(env, lp);
cur_numcols = CPXgetnumcols(env, lp);

x     = (double *)malloc(cur_numcols * sizeof(double));
slack = (double *)malloc(cur_numrows * sizeof(double));
dj    = (double *)malloc(cur_numcols * sizeof(double));
pi    = (double *)malloc(cur_numrows * sizeof(double));
```

Obter a solução

- A função **CPXsolution** retorna:
  - **solstat**: resultado da otimização.
  - **objval**: valor da função objetivo.
  - **x**: valores das variáveis primais.
  - **pi**: valores das variáveis duais.
  - **slack**: valores das variáveis de folga.
  - **dj**: valores dos custos relativos.

```
status = CPXsolution(env, lp, &solstat, &objval, x, pi, slack, dj);
```

Liberar memória

- Por fim, toda memória alocada deve ser liberada.
- Para destruir o objeto *problema* deve ser usada a função **CPXfreeprob**.
- O ambiente do CPLEX é finalizado chamando a função **CPXcloseCPLEX**.

```
free_and_null((char **)&x);
free_and_null((char **)&slack);
free_and_null((char **)&dj);
free_and_null((char **)&pi);

status = CPXfreeprob(env, &lp);
status = CPXcloseCPLEX(&env);
```

Construir o problema

- A **CPLEX Callable Library** também permite inserir os dados do problema chamando as funções **CPXnewcols** e **CPXaddrows**.

- Para isso é preciso primeiro determinar o número de linhas e colunas da matriz de restrição e o número de elementos não nulos.

  - Número de colunas: quantidade de variáveis da função objetivo.
  - Número de linhas: quantidade de restrições.
  - Número de não nulos: quantidade de variáveis presentes nas restrições.

Resolvendo PPL - CPLEX Callable Library - Exemplo 2

Dimensões dos arrays

```
Minimize x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10 + x11 + x12 + x13

Subject To
c1: 3x1 + 2x2 + x3 + 2x8 + 2x9 + x10 + x11 + x12 >= 4
c2: 2x4 + x5 + x8 + 2x10 + x11 >= 3
c3: 2x6 + x7 + x9 + x12 + x13 >= 2

Bounds
0 <= x1 <= 3
0 <= x2 <= 2
0 <= x3 <= 2

End
```

- Neste exemplo:

```
#define NUMROWS   3
#define NUMCOLS   13
#define NUMNZ     18
```

Colunas do problema

- O próximo passo é criar as colunas do problema. Para isso, é necessário preencher os arrays: **obj**, **lb**, **ub** e **colname.**
  - **obj**: coeficientes da função objetivo.
  - **lb**: limitante inferior.
  - **ub**: limitante superior.
  - **colname**: nome das colunas.

Colunas do problema

- O array **obj** deve ser preenchido com os coeficientes das variáveis da função objetivo.

```
double obj[NUMCOLS] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
```

```
Minimize x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10 + x11 + x12 + x13
```

Colunas do problema

- Os arrays **lb** e **ub** devem ser preenchidos com os coeficientes dos limitantes inferior e superior das variáveis de decisão, respectivamente.

```
double lb[NUMCOLS] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
double ub[NUMCOLS] = { 3, 2, 2, CPX_INFBOUND, CPX_INFBOUND, CPX_INFBOUND,
                       CPX_INFBOUND, CPX_INFBOUND, CPX_INFBOUND, CPX_INFBOUND,
                       CPX_INFBOUND, CPX_INFBOUND, CPX_INFBOUND };
```

```
Bounds
0 <= x1 <= 3
0 <= x2 <= 2
0 <= x3 <= 2
```

Colunas do problema

- Os arrays **lb** e **ub** devem ser preenchidos com os coeficientes dos limitantes inferior e superior das variáveis de decisão, respectivamente.

```
double lb[NUMCOLS] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
double ub[NUMCOLS] = { 3, 2, 2, CPX_INFBOUND, CPX_INFBOUND, CPX_INFBOUND,
                       CPX_INFBOUND, CPX_INFBOUND, CPX_INFBOUND, CPX_INFBOUND,
                       CPX_INFBOUND, CPX_INFBOUND, CPX_INFBOUND };
```

- **OBSERVAÇÕES:** Para as variáveis que não possuem limitantes, devemos preencher com zero o limitante inferior e CPX_INFBOUND o limitante superior.

Colunas do problema

- O array **colname** deve ser preenchido com os nomes das variáveis.

```
char *colname[NUMCOLS] = { "x1", "x2", "x3", "x4", "x5", "x6", "x7", "x8",
                           "x9", "x10", "x11", "x12", "x13" };
```

Colunas do problema

- Após preencher os arrays, é preciso chamar a função **CPXnewcols** para criar as colunas no objeto *problema*.

```
status = CPXnewcols(env, lp, NUMCOLS, obj, lb, ub, NULL, colname);
```

Linhas do problema

- Para criar as linhas os arrays **sense**, **rhs**, **rmatbeg**, **rmatind**, **rmatval** e **rowname** devem ser preenchidos.
  - **rhs**: vetor dos termos independentes.
  - **sense**: tipo da restrição.
  - **rmatbeg**: índices do primeiro elemento de cada linha.
  - **rmatind**: índices das variáveis.
  - **rmatval**: valores das variáveis.
  - **rowname**: nome das restrições.

Linhas do problema

- O array **rhs** deve ser preenchido com o vetor dos termos independentes.

```
double rhs[NUMROWS] = { 4, 3, 2 };
```

```
Subject To
c1: 3x1 + 2x2 + x3 + 2x8 + 2x9 + x10 + x11 + x12 >= 4
c2: 2x4 + x5 + x8 + 2x10 + x11 >= 3
c3: 2x6 + x7 + x9 + x12 + x13 >= 2
```

Resolvendo PPL - CPLEX Callable Library - Exemplo 2

Linhas do problema

- O array **sense** deve ser preenchido com o tipo da restrição.

```
char sense[NUMROWS] = { 'G', 'G', 'G' };
```

L	restrição de <=
E	restrição de =
G	restrição de >=

```
Subject To
c1: 3x1 + 2x2 + x3 + 2x8 + 2x9 + x10 + x11 + x12 >= 4
c2: 2x4 + x5 + x8 + 2x10 + x11 >= 3
c3: 2x6 + x7 + x9 + x12 + x13 >= 2
```

Linhas do problema

- O array **rmatind** e **rmatval** devem ser preenchidos com os índices e valores das variáveis de decisão, respectivamente.

```
int rmatind[NUMNZ] = { 0, 1, 2, 7, 8, 9, 10, 11, 3, 4, 7, 9, 10, 5, 6, 8, 11, 12 };
double rmatval[NUMNZ] = { 3, 2, 1, 2, 2, 1, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1 };
```

```
Subject To
c1: 3x1 + 2x2 + x3 + 2x8 + 2x9 + x10 + x11 + x12 >= 4
c2: 2x4 + x5 + x8 + 2x10 + x11 >= 3
c3: 2x6 + x7 + x9 + x12 + x13 >= 2
```

Linhas do problema

- O array **rmatbeg** deve ser preenchido com os índices do primeiro elemento de cada linha.

```
int rmatbeg[NUMROWS] = { 0, 8, 13 };
```

```
Subject To
c1: 3x1 + 2x2 + x3 + 2x8 + 2x9 + x10 + x11 + x12 >= 4
c2: 2x4 + x5 + x8 + 2x10 + x11 >= 3
c3: 2x6 + x7 + x9 + x12 + x13 >= 2
```

Linhas do problema

- O array **rowname** deve ser preenchido com os nomes das restrições.

```
char *rowname[NUMROWS] = { "c1", "c2", "c3" };
```

Linhas do problema

- Para criar as novas linhas no objeto *problema*, a função **CPXaddrows** deve ser chamada.

```
status = CPXaddrows(env, lp, 0, NUMROWS, NUMNZ, rhs, sense, rmatbeg,
                    rmatind, rmatval, NULL, rowname);
```

Função objetivo

- Para finalizar, deve-se informar se o problema é de minimização ou maximização chamando a função **CPXchgobjsen**.

```
status = CPXchgobjsen(env, lp, CPX_MIN);
```

CPX_MIN	para problemas de minimização
CPX_MAX	para problemas de maximização

Escrever o problema em arquivo

- A função **CPXwriteprob** escreve o problema criado em um arquivo, o qual pode ser utilizado para conferir se o problema foi construído corretamente.

```
status = CPXwriteprob(env, lp, "exemplo2.lp", NULL);
```

Exercício

- Uma empresa de navegação deseja embarcar um container com 9 tipos de itens de forma maximizar os custos da carga que poderá comercializar. Existe uma restrição de espaço de 1000 pés cúbicos e um limite de peso de 1200 libras.
- As dimensões dos tipos de itens a serem embarcados, os pesos e os custos de cada um deles são dados nos vetores *w, p, v*, respectivamente.
  - $w = (774\ \ 76\ \ 22\ \ 42\ \ 21\ \ 760\ \ 818\ \ 62\ \ 785)^T$.
  - $p = (\ 67\ \ 27\ \ 794\ \ 53\ \ 234\ \ 32\ \ 792\ \ 97\ \ 435)^T$.
  - $v = (77\ \ 6\ \ 3\ \ 6\ \ 33\ \ 13\ \ 110\ \ 21\ \ 47)^T$.

(a) Modele matematicamente o problema como um problema da mochila inteiro.

(b) Resolva o modelo pelo prompt do CPLEX.

(c) Resolva o modelo usando Callable library.

LIMEIRA 2019

51º SBPO

Obrigada!

Kelly Poldi
kelly@ime.unicamp.br

Carla Ghidini
carla.ghidini@fca.unicamp.br

FAPESP