外部接続マニュアル

Windows（VC++・BC++）版，UNIX/Linux 版

株式会社 数理システム
Phone: 03-3358-1701
Fax: 03-3358-1727
Email: nuopt-support@msi.co.jp

2009/03/10 更新

目次

1. はじめに

このドキュメントでは，簡単なデモンストレーションを通じて，NUOPT と C++による外部プログラムの連結方法を解説します．Windows 版では，VisualC++の GUI[1]を利用した例を用いて行います[2]．(Microsoft Visual C++.net をお使いのユーザは「6.3 節 Microsoft Visual C++.net プロジェクトの設定」と合わせてお読みください．）外部プログラムとの連結方法には大きく分けて，

1. ライブラリ solveLP,solveQP をコールする
2. SIMPLE のモデル記述からクラスを生成して利用する
3. SIMPLE のモデル記述を手続きの中に記述して利用する

という三つの方法があります．このドキュメントではこれらについて順に解説します．

➢ VC++

フォルダ：

(NUOPT のインストール場所）¥samples¥app

(デフォルトの NUOPT のインストール場所は c:¥Program File¥NUOPT)

の下に VC++のソリューションである

nuoptvcapp.sln

があります．まずこれを開いてください．このソリューション内には以下の４つのプロジェクトがあります．

これらのプロジェクトが以下の章で説明する実行例に対応しています．これらの実行例で用いるのは簡単な整数計画問題である次のナップサック問題です．

---

[1] Microsoft Visual Studio 2008 に基づいて説明しております．VisualC++.NET2002,.NET2003 用のサンプル御所望の方は nuopt-support@msi.co.jp までご連絡ください．

[2] BC++コンパイラで同様のことも可能ですが，パッケージにサンプルは納められていません．ご所望の方は nuopt-support@msi.co.jp までご連絡ください．

変数	$x_i \in \{0,1\} \quad (i \in S)$
目的関数（最大化）	$\sum_{i \in S} c_i x_i,$
制約条件	$\sum_{i \in S} a_i x_i \leq b$

S の要素数だけ０－１変数があり，線形制約が一本，線形の目的関数を最大化するという問題です．この問題を設定するのに必要なデータは，

目的関数の係数 c_i，制約式の係数 a_i，制約式の右辺値 b

となります．この問題を解くというアプリケーションを上記の様々な方法で行います．

1.1 サポートプラットフォーム

このマニュアルに記述されているインタフェースの内容と連結方法は Windows 版，UNIX/Linux 版共通です. ただし，UNIX/Linux 版の以下のプラットフォームでは，上記 2. の モデル記述からクラスを生成して利用する方法 はサポートされておりませんのでご了承ください.

HP-UX	HP-UX9.x, HP-UX10.2,HP-UX11.0

1.2 実行環境に含まれるもの

外部プログラムの連結を行うための実行環境には次が含まれます. 一部環境によって必要なファイルが異なります.

- インクルードファイル（共通して必要）:

userapp/include/*.h	NUOPT のインクルードファイル

- VC++版ライブラリとサンプル（VC++でのリンクにのみ必要）:

userapp/lib/libnuopt_M*.lib	NUOPT ライブラリ（VC++版）
SAMPLES/app/	VC++版用サンプルディレクトリ

- BC++版ライブラリとサンプル（BC++でのリンクにのみ必要）:

userapp/lib/libnuopt_bcc.lib	NUOPT ライブラリ（BC++版）
	BC++版用サンプルディレクトリ[3]

- UNIX/Linux 版ライブラリとサンプル（UNIX/Linux 環境でのリンクにのみ必要）:

libnuopt/libnuopt.a	NUOPT ライブラリ（UNIX/Linux 版）
userapp/	UNIX/Linux 版用サンプルディレクトリ

このライブラリから生成したロードモジュールは NUOPT がインストールされている環境でのみ動作し，動作条件は，NUOPT に準じます.

[3] パッケージにサンプルは納められていません. ご所望の方は nuopt-support@msi.co.jp までご連絡ください.

1.3 サンプルディレクトリの内容

外部ライブラリの利用サンプルディレクトリです．以下がその一覧です．

- **VC++**

nuoptvcapp.sln	VC++ のソリューション
useSolveQP.cpp	solveLP,solveQP を用いるサンプルメイン
useSolveQP/qp312.txt	useSolveQP.cpp 用データ 1
useSolveQP/qp312I.txt	useSolveQP.cpp 用データ 2
useSolveQP/knapsack.txt	useSolveQP.cpp 用データ 3
knapsack.smp	クラスインターフェーステストモデル
main.cpp	knapsackSolve のメイン
driver.cpp	knapsackSolve のドライバ
dllmain.cpp	dll のメイン

- **BC++･UNIX**

makefile	サンプルの makefile
cflags.cfg	コンパイル用 CFLAGS の設定ファイル(BC++版のみ)
useSolveQP.cc	solveLP,solveQP を用いるサンプルメイン
qp312.txt	useSolveQP.cc 用データ１
knapsack.txt	useSolveQP.cc 用データ２
useClass.cc	クラスを用いるサンプルメイン
QP.smp	クラスインターフェーステストモデル１
qp312.dat	QP.smp 用データ
knapsack.smp	クラスインターフェーステストモデル２
useSimple.cc	SIMPLE のモデル記述を手続きの中に 直接記述する場合のサンプルメイン
MIP.cc	SIMPLE のモデル記述を手続きの中に 直接記述する場合の問題記述の例

➢ BC++

BorlandC++をデフォルト（C:¥Borland¥bcc55）以外のパスに設定されている場合にはmakefile の中の先頭の BCCPATH に BC++のインストール箇所を設定してください.
makefile の先頭部分は出荷時には以下のようになっています.

```
# BorlandC++ のインストール場所. 環境に応じて調整が必要
BCCPATH = c:¥Borland¥bcc55
```

2．ライブラリ **solveLP,solveQP**

solveLP,solveQP はそれぞれ（混合整数）線形計画問題,（混合整数）二次計画問題・二次制約付き二次計画問題を対象とする C++のライブラリです[4]．問題は次の形式に定式化されているものとして,変数や制約式の上下限や係数行列を C++の配列として直接引数として取ります．入力の際にモデリング言語は用いません．

- （混合整数）線形計画問題（solveLP が対応）

最小化・最大化	$\sum_j c_j \cdot x_j$	$j = 1, \cdots, m$
条件	$cu_i \geq \sum_j A_{i,j} \cdot x_j \geq cl_i$	$i = 1, \cdots, n$ $j = 1, \cdots, m$
	$bu_j \geq x_j \geq bl_j$	$j = 1, \cdots, m$
	（$x_j \in Z$）	（$j \in I$）

[4] solveQP は混合整数の二次制約付きの問題は扱うことができません．

- （混合整数）二次計画問題・二次制約付き問題(`solveQP` が対応)

最小化・最大化	$\sum_j c_j \cdot x_j + \frac{1}{2}\sum_{j,k} Q_{j,k} \cdot x_j \cdot x_k$	$j = 1, \cdots, m$ $k = 1, \cdots, m$
条件	$cu_i \geq \sum_j A_{i,j} \cdot x_j + \frac{1}{2}\sum_{j,k} Q^i_{j,k} \cdot x_j \cdot x_k \geq cl_i$	$i = 1, \cdots, n$ $j = 1, \cdots, m$
	$bu_j \geq x_j \geq bl_j$	$j = 1, \cdots, m$
	（$x_j \in Z$）	（$j \in I$）

2.1 呼び出し形式

以下がライブラリ関数 solveLP,solveQP の呼び出し形式です．"=0"は C++の記法のデフォルト引数（省略が可能な引数）を示しています．solveLP は整数変数が存在しないとき，ivtype 以降を，solveQP は制約式の二次の部分が存在しないとき，nQCelem 以降をそれぞれ一括して省略することができます．呼び出し形式はヘッダーファイル

nuoIf.h

に含まれています．このヘッダーファイルは NUOPT ライブラリの include ファイルの保管場所

(Windows 版)：

（NUOPT のインストール場所）¥userapp¥include

(UNIX 版)：

（NUOPT のインストール場所）/userapp/include

にあります．

◆ solveLP

```
nuoptResult*
solveLP
(
 nuoptParam* options
 ,int n,int m
,int minimize
 ,double* x0
 ,double* bL,double* bU,int* ibL,int* ibU
 ,double* cL,double* cU,int* icL,int* icU
 ,double* objL
 ,int nAelem,int* irowA,int* jcolA,double* a
,int* ivtype = 0
 ,int* pri  = 0
 ,int* dir    = 0
 ,int* until  = 0
 ,double* upc      = 0
 ,double* dpc    = 0
);
```

一括して省略可能（整数変数が存在しないとき）

◆ solveQP

```
nuoptResult*
solveQP
(
 nuoptParam* options
 ,int n,int m
,int minimize
 ,double* x0
 ,double* bL,double* bU,int* ibL,int* ibU
 ,double* cL,double* cU,int* icL,int* icU
 ,double* objL
 ,int nAelem,int* irowA,int* jcolA,double* a
 ,int nQelem,int* irowQ,int* jcolQ,double* q
 ,int nQCelem = 0 ,int* ifunQC = 0,int* irowQC = 0 ,int* jcolQC = 0
 ,double* qc = 0
  ,int* ivtype = 0
 ,int* pri  = 0
 ,int* dir    = 0
 ,int* until  = 0
 ,double* upc      = 0
 ,double* dpc    = 0
);
```

一括して省略可能（制約式の二次の部分が存在しないとき）

一括して省略可能（整数変数が存在しないとき）

solveQP は solveLP の機能を含んでおり，solveQP で二次の項の指定をすべて消去すると，solveLP と同一の機能となります.

2.2 ルーチン仕様

solveLP,solveQP の引数の意味は共通しています．以降で各引数の解説を行います．

2.2.1 問題全体にかかわるもの(solveLP,solveQP 共通)

nuoptParam*	options	NUOPT の求解制御パラメータ（0：デフォルト）
int	n	変数の総数
int	m	制約式の総数（等式，不等式含む）
int	minimize	最小化かどうかのフラグ（零以外：最小化，0：最大化）
double*	x0	変数の初期値（長さ：n）
double*	bL	変数の下限ベクトル（長さ：n）
double*	bU	変数の上限ベクトル（長さ：n）
int*	ibL	変数の下限の有無（長さ：n，非零：あり，0：なし）
int*	ibU	変数の上限の有無（長さ：n，非零：あり，0：なし）
double*	cL	制約式の下限ベクトル（長さ：m）
double*	cU	制約式の上限ベクトル（長さ：m）
int*	icL	制約式の下限の有無（長さ：m，非零：あり，0：なし）
int*	icU	制約式の上限の有無（長さ：m，非零：あり，0：なし）

ここで紹介する引数は solveLP,solveQP に共通で，同じ意味を持ちます．最初の引数は NUOPT の求解制御パラメータを与えるものですが，0 を渡すことができます．その場合にはパラメータとしてデフォルトの設定を用いるという意味になります.通常はデフォルトで問題ありませんが，パラメータの指定を行う場合には，

```
nuoptParam myparam;
myparam.method = “asqp”; // 求解メソッドの指定
solveQP(&myparam,…);
```

のようにして，myparam のメンバ（個別のパラメータに相当する）に値を設定して，solveQP にアドレスを渡します．詳細は「5.7NUOPT オプション」を参照ください．

変数の上下限が存在しない場合には，対応する ibL または ibU のコンポーネントを 0 にします．例えば n = 3 の場合で

$$0 \le x_1$$
$$x_2 \le 1$$
$$2 \le x_3 \le 3$$

という上下限を表現する場合には次のように設定します．

配列の添字	bL	bU	ibL	ibU
0	0	任意	0 以外	0
1	任意	1	0	0 以外
2	2	3	0 以外	0 以外

C++では配列の添字は 0 はじまりですので，1 番目の変数は添字 0 の場所に対応します．配列の「任意」と書かれた場所は無視されます．「0 以外」という場所は 0 以外の任意の値です．制約式についても全く同様です．

変数の固定，および等式制約は上限と下限を一致させることによって表現します．ibL,ibU,icL,icU の場所に NULL ポインタを渡すことが許されています．その場合，すべて 0 が格納された配列を渡すのと同じ意味になります．

初期値に対応する x0 に NULL ポインタを渡すことができます．その場合には初期値はデフォルトのものを用いるという意味になります．

2.2.2 線形部分にかかわるもの(solveLP,solveQP 共通)

double*	objL	目的関数の線形部分（長さ：n）
int	nAelem	制約式の係数行列の非零要素数
int*	irowA	非零要素の行番号（長さ：nAelem）
int*	jcolA	非零要素の列番号（長さ：nAelem）
double*	a	非零要素の値（長さ：nAelem）

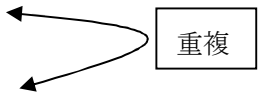
ここで扱う引数も solveLP,solveQP で同じ意味を持ちます．objL は目的関数の線形部分

$$\sum_j c_j \cdot x_j$$

の c_j に対応します．

制約式の係数行列は非零要素のみ与えます．同じ非零要素を二つ以上与えた場合には，それらの和が取られます．非零要素の場所は 1 始まりの番号で与えます．例えば nAelem=4 のとき，

配列の添字	irowA	jcolA	a
0	1	2	2.3
1	1	3	1.8
2	2	1	0.5
3	1	3	0.2



のように与えると

$$A = \begin{bmatrix} 0 & 2.3 & 2.0 \\ 0.5 & 0 & 0 \end{bmatrix}$$

を与えたことになります（A_{13} については二つ与えられているので和（1.8+0.2=2.0）が取られます)．非零要素の格納順番は任意です．

2.2.3 混合整数計画問題にかかわるもの(solveLP, solveQP 共通：一括省略可)

```
int*     ivtype   変数の種別
                  （長さ：n，0:連続，1：整数，2：バイナリ，3：準連続，4：部分連続）
int*     pri      変数の分枝優先度
                  （長さ：n，0 より小さければ無視，小さい値のものほど優先）
int*     dir      変数の優先される分枝方向
                  （長さ：n，正の値：押し上げ，負の値：押し下げ，0：指定なし）
int* until   部分連続変数の境界値（長さ：n）
double*  upc      変数の押し上げ擬コスト（長さ：n，負なら無視，正の値）
double*  dpc      変数の押し下げ擬コスト（長さ：n，負なら無視，正の値）
```

ここに挙げた引数で整数変数の指定を行います．ivtype が変数の種別を与えるベクトルで，次の意味を持ちます．

0：連続変数
1：一般の整数変数
2：バイナリ変数（0 または 1）
3：準連続変数（0 たは 1 上の実数）
4：until で与えられた値を超えるまで整数値をとり，それ以上は実数

ivtype 自体を NULL ポインタにするとすべての変数が連続変数であるという指定と同じ意味となります．until を NULL ポインタにするとすべて 0 を与えたのと同じ意味になります．

pri,upc,dpc,dir はモデリング言語 SIMPLE を用いた場合に IntegerVariable のメンバとして与えられるものと同じで以下の意味を持ちます．

- pri (分枝優先順位) の値は小さいものが先に分枝されることになります．分枝優先順位の与えられている変数と与えられていない変数が混在している場合には，分枝優先順位の与えられていない変数は最低の優先順位が与えられたと見なされます． 優先順位として 0 以下の値は与えられても無視し， 優先順が与えられないのと同じ扱いとします．
- upc,dpc(擬コスト) の与えられているものと与えられていないものが混在する場合,擬コストが与えられていない変数については擬コストがすべて零であるものとします． どの変数にも擬コストが全く与えられていない場合にのみ，NUOPT は擬コストを緩和問題から推定した値に設定します．負の擬コストは無視し， 与えられていないのと同じ扱いとします．
- dir(分枝優先方向)は最初に分枝する方向に関してのパラメータで，符号のみが問題になります．正なら押し上げを優先，負なら押し下げを優先，0 なら擬コストからの推定を用いて NUOPT がどちらを優先するかを決定します．

pri,dir, upc,dpc はすべて NULL ポインタとすることができます．その場合にはこれらに

ついて指定をしないのと同じ意味となります.
整数変数が存在しない場合には，ivtype から以降の引数をすべて省略することができます（一括省略）.

2.2.4 目的関数の二次の部分にかかわるもの(solveQP のみ)

int	nQelem	目的関数のヘッセ行列の非零要素数
int*	irowQ	目的関数のヘッセ行列の行番号（長さ：nQelem）
int*	jcolQ	目的関数のヘッセ行列の列番号（長さ：nQelem）
double*	q	目的関数のヘッセ行列の非零要素の値（長さ：nQelem）

ここに挙げた引数で目的関数の二次の部分の係数行列（ヘッセ行列）

$$\frac{1}{2}\sum_{j,k} Q_{j,k} \cdot x_j \cdot x_k$$

の $Q_{i,j}$ を与えます. 制約式の係数行列と同様に非零要素のみを与えます. 指定の仕方は制約式の係数行列と同様です．非零要素の場所は 1 始まりの番号で与えます．しかし，目的関数のヘッセ行列は対称行列であることを前提としていますので, 下三角部分の非零要素を与えると同時に上三角部分も与えたことになる（その逆も同じ）という点に注意してください.
例えば

$$Q = \begin{bmatrix} 1 & 5 \\ 5 & 7 \end{bmatrix}$$

というヘッセ行列を定義するためには nQelem = 3 として

配列の添字	irowQ	jcolQ	q
0	1	1	1
1	2	1	5
2	2	2	7

→ 同時に上三角側も与えたことになる.

のように与えます.
あるいは以下のようにしても同じ意味です.

配列の添字	irowQ	jcolQ	q
0	1	1	1
1	1	2	5
2	2	2	7

→ 同時に下三角側も与えたことになる.

同じ非零要素を二つ与えるとその和が取られます．従って nQelem=4 として

配列の添字	irowQ	jcolQ	q
0	1	1	1
1	1	2	5
2	2	1	5

→ 二倍にカウントされてしまう.

```
3              2          2          7
```

とすると,上の下三角部分の非零要素を与えると同時に上三角部分も与えたことになるという原則（その逆も同じ）によって

$$Q=\begin{bmatrix}1 & 10\\ 10 & 7\end{bmatrix}$$

を定義したことになりますのでご注意ください．また，非零要素の格納順番は任意です．

nQelem=0 とすると，目的関数に二次の部分が存在しないものと解釈されます．その場合には irowQ,jcolQ,q はすべて NULL ポインタとすることができます.

2.2.5 制約式の二次の部分にかかわるもの(solveQP のみ，一括省略可)

int	nQCelem	制約式のヘッセ行列の非零要素数
int*	ifunQC	制約式のヘッセ行列の非零要素が属する制約式番号（長さ：nQCelem）
int*	irowQC	制約式のヘッセ行列の行番号（長さ：nQCelem）
int*	jcolQC	制約式のヘッセ行列の列番号（長さ：nQCelem）
double*	qc	制約式のヘッセ行列の非零要素の値（長さ：nQCelem）

これらの引数で制約の二次部分

$$\frac{1}{2}\sum_{j,k}Q^{i}_{j,k}\cdot x_j\cdot x_k \qquad (i\text{は制約式の番号})$$

の係数行列（ヘッセ行列）の群

$$Q^{i}_{j,k}$$

を与えます．非零要素と，その非零要素が属する制約式の番号のみを与えます.

例えば，制約式 1，2 の二次の部分がそれぞれ

$$Q^1=\begin{bmatrix}4 & 3\\ 3 & 6\end{bmatrix},\ Q^2=\begin{bmatrix}8 & 2\\ 2 & 3\end{bmatrix}$$

である場合には nQCelem = 6 として

配列の添字	ifunQC	irowQC	jcolQC	qc
0	1	1	1	4
1	1	1	2	3
2	1	2	2	6
3	2	1	1	8
4	2	1	2	2
5	2	2	2	3

と設定します．制約式は 1 始まりの番号で指定します．また，行列の非零要素の場所も 1 始まりの番号で指定します.

制約式のヘッセ行列も対称であることを前提としているので,目的関数のヘッセ行列と同じく下三角部分の非零要素を与えると同時に上三角部分も与えたことになる（その逆も同じ）という原則が適用されます．また，目的関数の二次部分のヘッセ行列と同じく，同一の非零要素が二つ以上与えられると和が取られます.

非零要素の格納順番は任意です.

nQCelem=0 とすると，制約式に二次の部分が存在しないものと解釈されます．その場合には ifunQC,irowQC,jcolQC,qc はすべて NULL ポインタとすることができます.

制約式の二次の部分が存在しない場合には，nQCelem から以降の引数をすべて省略することができます.

2.2.6 出力とエラーメッセージ

solveLP,solveQP の戻り値として，nuoptResult というオブジェクトのポインタが返ります．このオブジェクトのメンバに実行結果についての情報が格納されています. 利用可能なメンバは以下の通りです．このメンバの利用の詳細は次項で解説するサンプルコード useSolveQP.cpp(.cc) をご覧ください．このオブジェクトは利用するコード内で開放(delete)する必要があります.

int errorCode();	エラーコード
char* errorMessage();	エラーメッセージ
double optValue();	最適値
double VarVal(int i);	i 番目の変数の値
double VarDual(int i);	i 番目の変数の dual 値
double FuncVal(int i);	i 番目の制約の値
double FuncDual(int i);	i 番目の制約の dual 値

上記で i は変数，制約式のインデクスですが，0 始まりであることにご注意ください（最初の変数/制約式が 0 番目）.

エラーコードとエラーメッセージは NUOPT 本体のものと同じです.「NUOPT/SIMPLE マニュアル」の付録 A をご参照ください．また，solveLP,solveQP 特有のエラーとして，引数の矛盾や範囲オーバーがありますが，それは 101 以上のコード番号に対応します．意味は以下のとおりです.

コード	意味
101	n が負
102	m が負
103	変数の下限(bL)が上限(bU)よりも大きい
104	制約式の下限(cL)が上限(cU)よりも大きい
105	nAelem が負
106	irowA のコンポーネントの範囲が違反
107	jcolA のコンポーネントの範囲が違反

```
108     nQelem が負
109     irowQ のコンポーネントの範囲が違反
110     jcolQ のコンポーネントの範囲が違反
111     nQCelem が負
112     ifunQC のコンポーネントの範囲が違反
113     irowQC のコンポーネントの範囲が違反
114     jcolQC のコンポーネントの範囲が違反
115     ivtype の範囲が違反
```

エラーメッセージは

```
<<NUOPT 106>> irowA[0] = 9 should be in [1,3]
```

のように，実際のデータに即したより詳しい情報を含んでいます.

2.3 実行サンプル

開発環境と同梱されているサンプル useSolveQP.cpp(.cc)は solveLP,solveQP を利用するサンプルです. 線形計画問題なら, solveLP を, 二次計画問題なら, solveQP を呼びます.

➢ VC++

ソリューション nuoptvcapp のプロジェクト

useSolveQP

がこのインタフェースを用いて LP,QP を解くプログラム例です.

➢ Win・UNIX共通

useSolveQP.cpp(.cc)は solveLP,solveQP を利用するサンプルで，１つの main 関数のみから成ります. ロードモジュールの引数名で与えられたファイルから問題のデータを読み込み, それを solveLP,solveQP に渡します．二次の項がなく，線形計画問題ならば solveLP を，あれば solveQP を呼びます．以下はそのリストの抜粋です.

```
//
//  solveQP の利用例
//
#include "nuoIf.h" // 必須.
int main(int argc, char** argv)
{
        int n;
        int m;
        ifstream inputFile(argv[1]);
        inputFile >> n >> m ;

                (中略)
        nuoptResult* qpres = 0;
        nuoptParam myParam;
        if ( nQelem || nQCelem ) { // ２次の係数があるなら solveQP をコール
        qpres = solveQP(&myParam
        ,n,m
        ,minimize
        ,x0
        ,bL,bU,ibL,ibU
```

データ読み込み

ロードモジュールの引数をファイル名とする

求解

solveQP のコール

```
        ,cL,cU,icL,icU
        ,objL
        ,nAelem,irowA,jcolA,a
        ,nQelem,irowQ,jcolQ,q
        ,nQCelem,ifunQC,irowQC,jcolQC,qc
        ,ivtype,pri,dir,until,upc,dpc
        );
        } else {// ２次の係数がないのなら solveLP  をコール
        qpres = solveLP(&myParam
        ,n,m
        ,minimize
        ,x0
        ,bL,bU,ibL,ibU
        ,cL,cU,icL,icU
        ,objL
        ,nAelem,irowA,jcolA,a
        ,ivtype,pri,dir,until,upc,dpc
        );
}
```

solveLP のコール

エラーの場合

```
        if ( qpres->errorCode() ) {
               printf("error in solveQP code = %d, message = %s¥n"
               ,qpres->errorCode(),qpres->errorMessage());
        fflush(stdout);
        exit(1);
        } else {
               printf("optimalValue = %17.10e¥n",qpres->optValue());
               printf("X:¥n");
               for ( i = 0 ; i < n ; ++i ) {
               printf("[%3d] %10.3e ",i+1,qpres->VarVal(i));
               if ( (i+1) % 4 == 0 ) {
                      printf("¥n");
               }
        }
        printf("¥n");
        printf("F:¥n");
        for ( i = 0 ; i < m ; ++i ) {
               printf("[%3d] %10.3e ",i+1,qpres->FuncVal(i));
               if ( (i+1) % 4 == 0 ) {
                      printf("¥n");
               }
        }
               printf("¥n");
        }
```

解の表示

```
         delete qpres;
         delete [] bL;
         delete [] bU;
         delete [] ibL;
         delete [] ibU;
         (後略)
}
```

終了処理

次のようなナップサック問題を考えます.

変数　$x_i \in \{0,1\} \quad (i \in S)$

目的関数（最大化）　$\sum_{i \in S} c_i x_i,$

制約条件　$\sum_{i \in S} a_i x_i \le b$

目的関数の係数：　$c = (42 \quad 12 \quad 45 \quad 5 \quad 2 \quad 61 \quad 89 \quad 32 \quad 47 \quad 18)$

制約式の係数：　$a = (39 \quad 13 \quad 68 \quad 15 \quad 10 \quad 20 \quad 31 \quad 15 \quad 41 \quad 16)$

制約式の右辺：　$b = 121$

これを `solveLP` で解くには，一般の線形計画問題：

最小化・最大化　$\sum_j c_j \cdot x_j \qquad j = 1, \cdots, m$

条件　$cu_i \ge \sum_j A_{i,j} \cdot x_j \ge cl_i \qquad i = 1, \cdots, n \quad j = 1, \cdots, m$

$bu_j \ge x_j \ge bl_j \qquad j = 1, \cdots, m$

（$x_j \in Z \qquad j \in I$）

の形にこの問題を表現する，すなわち

$$c = (42 \quad 12 \quad 45 \quad 5 \quad 2 \quad 61 \quad 89 \quad 32 \quad 47 \quad 18)$$
$$Q = 0$$
$$cu = (121)$$
$$cl = (-\infty)$$
$$A = (39 \quad 13 \quad 68 \quad 15 \quad 10 \quad 20 \quad 31 \quad 15 \quad 41 \quad 16)$$

$x_j \in \{0,1\}$　　（バイナリ変数）

とすればよいことがわかります．useSolveQP.cpp(.cc)の入力ファイルとしてこのデータを表現したのがファイル

(VC++版)：

（NUOPT のインストール場所）¥samples¥app¥useSolveQP¥knapsack.txt

(BC++/UNIX 版)：

（NUOPT のインストール場所）/userapp/knapsack.txt

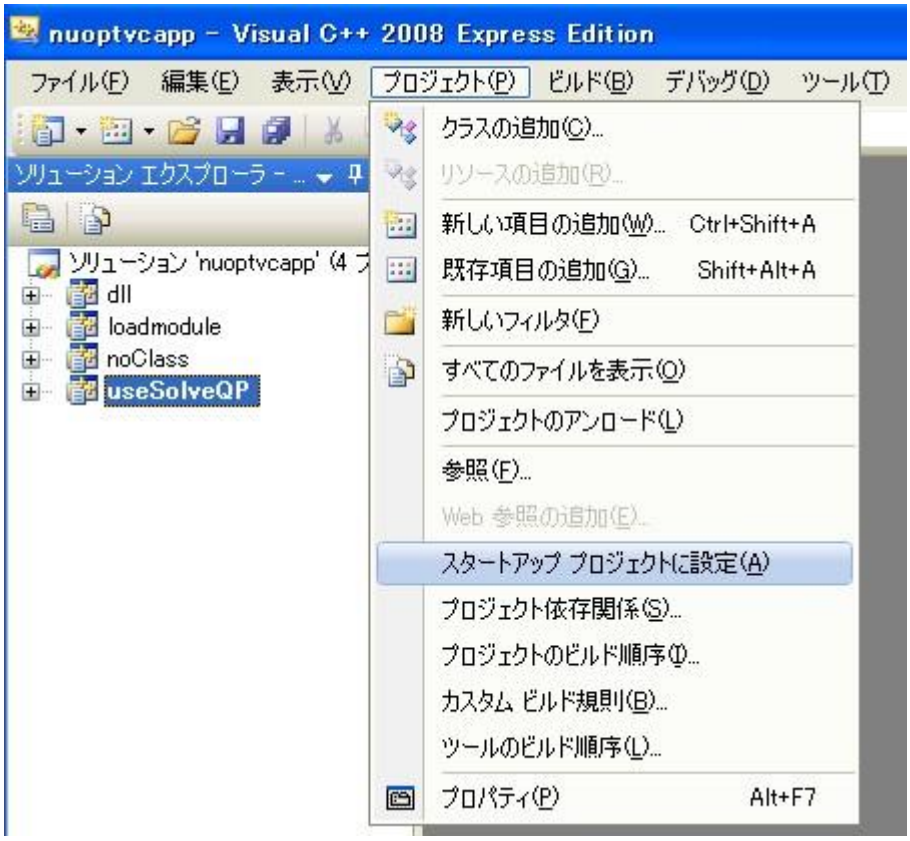
です．

2.4 実行例

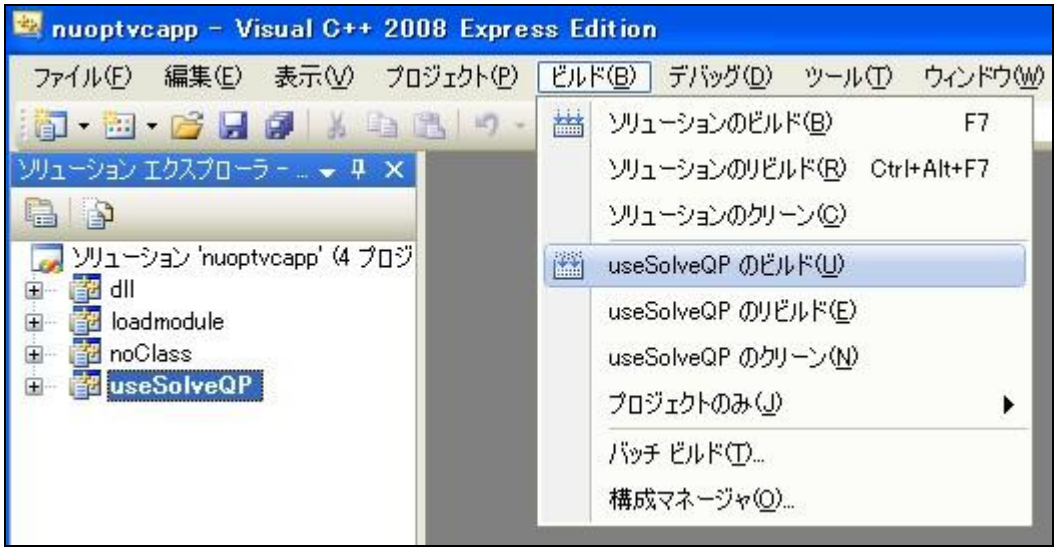
さて，サンプルコード useSolveQP の実行モジュールを作成して実行してみましょう．

- ***VC++***

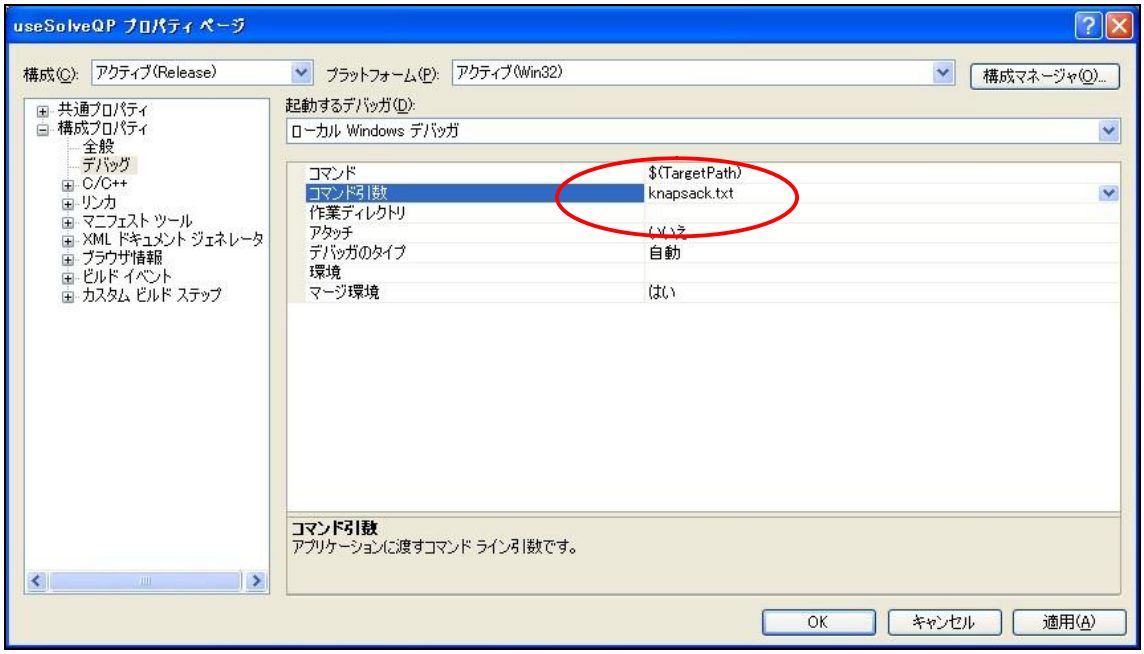
プロジェクト「 useSolveQP 」を選択後，VC++のメニューの「プロジェクト」から「スタートアップ プロジェクトに設定」を選び，プロジェクトを選択します．なお useSolveQP の構成は「Release 」に設定してあります．



続いて実行メニューのビルド：



を実行すると useSolveQP.cpp がコンパイル，NUOPT のライブラリとリンクされて，実行モジュールの useSolveQP.exe が作成されます. 引数として knapsack.txt を与えるように「プロジェクト」→「プロパティ」→「構成プロパティ」→「デバッグ」の項目で，「コマンド引数」に「 knapsack.txt」と入力します.



つづいて「デバッグ」メニューから

と実行すると，実行経過表示ウィンドウが現れ，その中に以下のような出力が得られます．solveLP がコールされて出力が成されます．

```
NUOPT x.x.x, Copyright (C) 1991-200x Mathematical Systems Inc.
PROBLEM_NAME                                        anon.LP
NUMBER_OF_VARIABLES                                      10
NUMBER_OF_FUNCTIONS                                       2
PROBLEM_TYPE                                   MAXIMIZATION
METHOD                                            SIMPLEX
<preprocess begin>.....<
<iteration begin>
       1.2B
    up:          244.71  lo:         -1e+050                          time:
0.0s:mem(Mb)=3/2:avail(Mb)=2029/647
                                                 llen:2 #prob:4 #piv:9
#1    up:          244.71  lo:              242                         time:
0.0s:mem(Mb)=3/2:avail(Mb)=2029/647
                                                 llen:2 #prob:5 #piv:10

<iteration end>
STATUS                                              OPTIMAL
VALUE_OF_OBJECTIVE                                      242
SIMPLEX_PIVOT_COUNT                                       7
PARTIAL_PROBLEM_COUNT                                    10
DUAL_SIMPLEX_PIVOT_COUNT                                  4
ELAPSED_TIME(sec.)                                     0.00
SOLUTION_FILE                                     solver.sol
optimalValue = 2.4200000000e+002
optimalValue = 2.4200000000e+002
X:
[  1] 1.000e+000 [  2] 0.000e+000 [  3] 0.000e+000 [  4] 0.000e+000
[  5] 0.000e+000 [  6] 1.000e+000 [  7] 1.000e+000 [  8] 1.000e+000
[  9] 0.000e+000 [ 10] 1.000e+0
F:
[  1] 1.210e+002
Press any key to continue
```

NUOPT が出す表示

useSolveQP が出す表示

上に挙げた内容は NUOPT からの求解に関する出力です．問題の変数の数（NUMBER_OF_VARIABLES）や目的関数の値（VALUE_OF_OBJECTVE）を知ることができます．

NUOPT の標準出力の内容については「NUOPT/SIMPLE マニュアル」（別冊）の第 2 部・2.4 節をご覧ください.

NUOPT の出力を抑制するには,

```
nuoptParam myParam; // 宣言
```

として，パラメータを宣言したのち,

```
myParam.outputMode = “silent”;
```

とします．また，デフォルトで出力される NUOPT の求解レポートである解ファイル solver.sol の出力を抑制するには

```
myParam.outfilename = “_NULL_”;
```

と設定し，solveQP()の最初の引数として渡します．useSolveQP.cpp(.cc)の以下のコメントを取ると，その設定になり，NUOPT からの出力が抑制されて，useSolveQP が出力する表示のみになります.

```
..
nuoptParam myParam;
// NUOPT の出力と解ファイルの出力を抑制する.
myParam.outputMode = "silent";
myParam.outfilename = "_NULL_";
  if ( nQelem || nQCelem ) { // ２次の係数があるのなら solveQP
  qpres = solveQP(&myParam
                  ,n,m
   ..
```

再びビルドメニューから

とすると，コンパイルが自動的に実行されます.

- **BC++･UNIX**

```
Prompt% make useSolveQP.exe
```

とすると，useSolveQP.cc がコンパイル，NUOPT のライブラリとリンクされて useSolveQP.exe が作成されます．引数として knapsack.txt を与えてロードモジュールを実行します.

次は，ナップサック問題に対応する knapsack.txt をこの設定で実行したときの出力です.

```
Prompt% useSolveQP.exe knapsack.txt
optimalValue = 2.4200000000e+002
X:
[  1] 1.000e+000 [  2] 0.000e+000 [  3] 0.000e+000 [  4] 0.000e+000
[  5] 0.000e+000 [  6] 1.000e+000 [  7] 1.000e+000 [  8] 1.000e+000
[  9] 0.000e+000 [ 10] 1.000e+000
F:
[  1] 1.210e+002
prompt%
```

- **Win･UNIX共通**

knapsack.txt と同じ場所に別の useSolveQP.cpp(.cc)の入力ファイル

(VC++版)：

（NUOPT のインストール場所）¥samples¥app¥useSolveQP¥qp312.txt

(BC++/UNIX 版)：

（NUOPT のインストール場所）/userapp/qp312.txt

があります．このデータは二次計画問題で

最小化・最大化 $\sum_j c_j \cdot x_j + \frac{1}{2}\sum_{(j,k)} Q_{j,k} \cdot x_j \cdot x_k$ $j = 1, \cdots, m$ $k = 1, \cdots, m$

条件 $cu_i \geq \sum_j A_{i,j} \cdot x_j + \frac{1}{2}\sum_{j,k} Q^i_{j,k} \cdot x_j \cdot x_k \geq cl_i$ $i = 1, \cdots, n$ $j = 1, \cdots, m$

$bu_j \geq x_j \geq bl_j$ $j = 1, \cdots, m$

（$x_j \in Z$ $j \in I$）

という定式で，次のように設定したものに対応します．2 変数，3 制約の問題で，制約式には二次の項はありません．

$$c = \begin{pmatrix} -3 & 1 \end{pmatrix}$$

$$Q = \begin{pmatrix} 11 & 0 \\ 0 & 22 \end{pmatrix}, \; Q^i = 0$$

$$cu = \begin{pmatrix} 1000 & 1000 & 1000 \end{pmatrix}$$

$$cl = \begin{pmatrix} -1 & -2 & 2 \end{pmatrix}$$

$$A = \begin{pmatrix} -1 & 0.1 \\ -0.2 & -1 \\ 2 & 1 \end{pmatrix}$$

$$bu = \begin{pmatrix} 1 & 2 \end{pmatrix}$$

$$bl = \begin{pmatrix} 0 & 0 \end{pmatrix}$$

(VC++版)：
VC++の GUI で「プロジェクト」→「設定」→「デバッグ」タブで，「プログラムの引数」を

```
qp312.txt
```

として再び実行してみてください．

(BC++/UNIX 版)：

```
prompt% useSolveQP.exe qp312.txt
```

とします.

今度は useSolveQP が solveQP をコールして，次のような解が出力されます.

```
optimalValue = 2.3181851911e+000
X:
[  1] 9.394e-001 [  2] 1.212e-001
F:
[  1] -9.273e-001 [  2] -3.091e-001 [  3] 2.000e+000
```

次のデータ(Windows 版のみ)：

（NUOPT のインストール場所）¥samples¥app¥useSolveQP¥qp312I.txt

この二次計画問題の変数を連続変数ではなく，整数変数という指定をつけたもので，二次の整数計画問題となります．これを入力として実行すると

```
optimalValue = 2.5000000000e+000
X:
[  1] 1.000e+000 [  2] 0.000e+000
F:
[  1] -1.000e+000 [  2] -2.000e-001 [  3] 2.000e+000
```

のような解が出力されます.

3．SIMPLE モデル記述からクラスを生成して利用する例

本章で紹介するのは SIMPLE によって記述したモデルを生かすことのできる方法です[5]．方法においては，まず数理計画モデルをモデリング言語 SIMPLE で記述します．次にその記述から数理計画モデルに対応する C++のクラスを生成します．ユーザプログラムではその C++のクラスのオブジェクトを宣言することによって

- データの読み込み
- 最適化の実行
- 変数の初期値の設定
- 最適解や関連する量の表示

などの操作を実現します．クラスオブジェクト同士は独立しているので，複数の問題をひとつの外部プログラムで操作するなどの操作が可能になります．

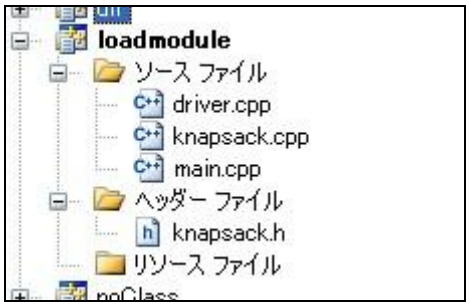
➢ VC++

本節では，SIMPLE のモデル記述から C++のクラスを生成して，ナップサック問題のモデルに対して操作を行う手続き（ドライバ）を作成する例を紹介します．

ソリューション nuoptvcapp のプロジェクト

loadmodule

がこのインタフェースを用いてナップサックを解くプログラム例です．



➢ BC++･UNIX

BC++/UNIX 版では，SIMPLE のモデル記述から C++のクラスを生成して，二次計画問題，ナップサック問題のモデルに対して操作を行う手続き（ドライバ）を作成する例を紹介します．

[5] このインタフェースは，「1.1 サポートプラットフォーム」で挙げられている一部の UNIX/Linux 版のプラットフォームではサポートされておりません．SIMPLE によって記述したモデル生かすには 4.2BC++/UNIX 版ライブラリ (SIMPLE モデルとドライバを分離しない例) をご利用ください．

ファイル

（NUOPT のインストール場所）/userapp/useClass.cc
/QP.smp
/knapsack.smp

がこのインタフェースを用いて二次計画問題，ナップサック問題を解くプログラム例です.

VC++版ライブラリをお使いの方は
3.1VC++版ライブラリ(SIMPLE モデル記述からクラスを生成して利用する例)へ
BC++/UNIX 版ライブラリをお使いの方は
3.2BC++/UNIX 版ライブラリ(SIMPLE モデル記述からクラスを生成して利用する例)へ
お進みください.

3.1 VC++版ライブラリ(SIMPLE モデル記述からクラスを生成して利用する例)

3.1.1 モデル VC++

次に示すプログラムはこのアプリケーションの核心となるモデル記述です. 通常のモデルと異なるのは, パラメータの宣言部分に required というキーワードがあることですが, これはこの指定のあったデータを C++の配列から入力することを示しています. その他は通常のモデル記述と同じです. この例ではこのモデルが次のような knapsack.smp というファイルに記述されたものとします.

```
//
//   ナップサック問題
//
//
    Set S;
    Element i(set=S);
    IntegerVariable x(index=i,type=binary); // 整数変数
    Parameter c(index=i,required);
    Parameter a(index=i,required);
    Parameter b(required);
    Objective obj(type=maximize);
    VariableParameter d(index=i);

    obj = sum(c[i]*x[i],i);    // 目的関数
    sum(a[i]*x[i],i) <= b;     // 制約条件
```

このファイルは本章で扱っている NUOPT の例を納めたソリューションと同じ場所

（NUOPT のインストール場所）¥samples¥app

にあります.

3.1.2 genClass のコール VC++

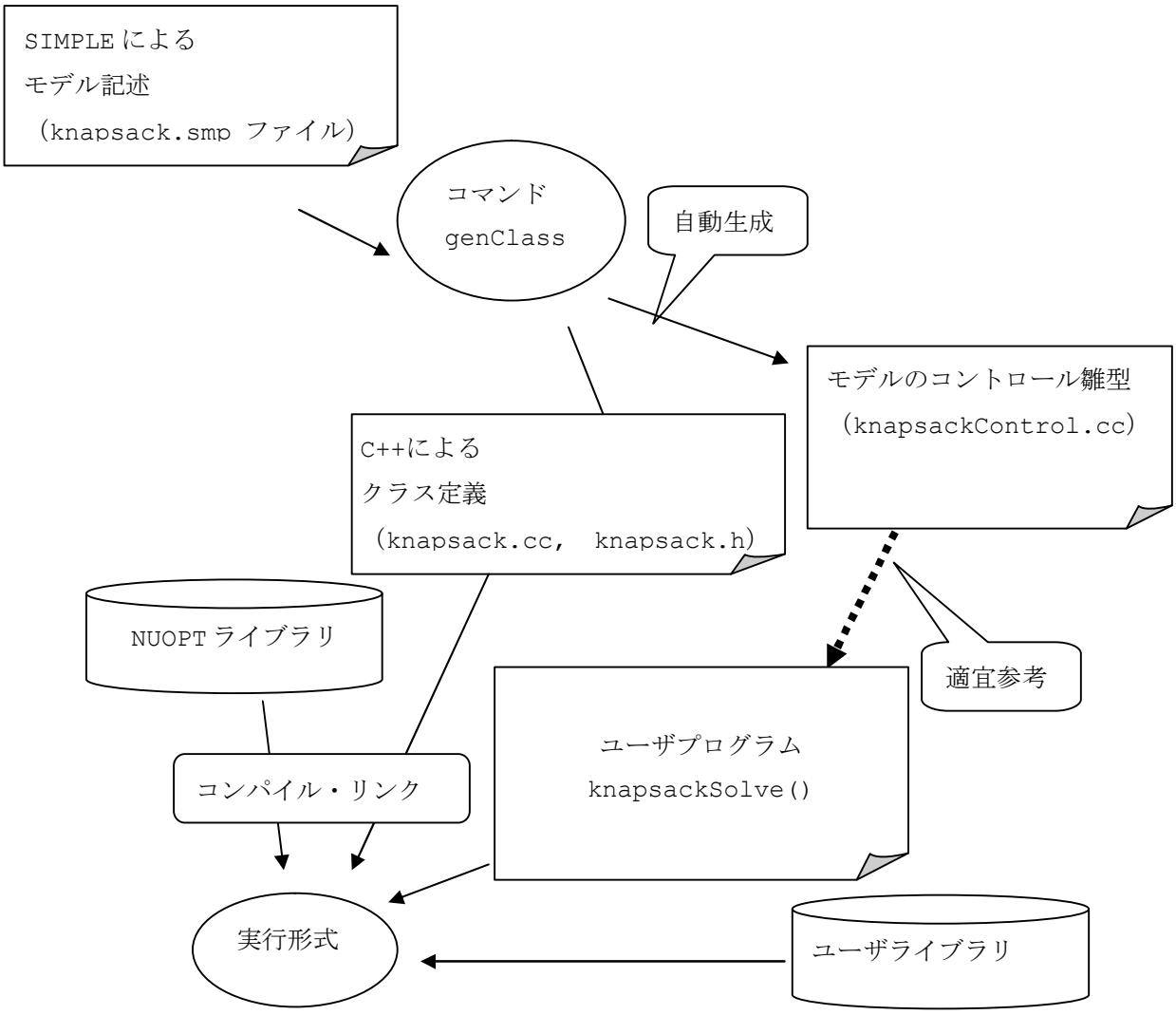
前節のモデルはこのままでは C++で利用できる形ではありませんが. genClass というコマンドを用いて 変換すると, 対応するクラス宣言（knapsack.h）とクラスの実装（knapsack.cc）が自動生成されます.

次のページに関連するファイルの関係を図示します. ユーザはまず, モデリング言語 SIMPLE を用いてモデル記述を行い, genClass によってクラスの実装[6]を自動生成し, ユーザプログラムからそれを使います. モデルを利用する場合には, こうして生成されたクラス宣言（同

[6] このクラスの実装の内容をユーザは直接意識する必要はありません.

knapsack.h）をインクルードして，クラスオブジェクトを宣言します．このクラスオブジェクトのメンバを通じて，ユーザは自分の C++プログラム（同 knapsackSolve()）からこのモデルにデータや解を入出力したり(print(),dump())する操作が可能になります．

実行形式の生成の際には，先ほど自動生成されたクラス実装をコンパイル，リンクします．こうして生成された SIMPLE のクラスは同時に何個でも,また何回でも利用することができます．ただし SIMPLE のクラスを利用するに先立って，ユーザのプログラムから処理前に SimpleInitialize()，処理後に SimpleClearBuffer()なる関数を呼び出す必要があります（これらは SIMPLE が独自に利用する記憶領域の確保と開放に必要となります)．



genClass の起動は DOS ウインドウのプロンプトから

```
> genClass knapsack.smp
```

とします．genClass は NUOPT が提供するユーティリティで，この実行のためには別途設定が必要となります．詳細については NUOPT Windows ヘルプにある「コマンドラインインタフェースの使い方」を参照してください．
以下が knapsack.smp に対する genClass の実行例です．

```
C:¥Program Files¥NUOPT¥SAMPLES¥app>genClass knapsack.smp
genClass for Windows(Ver. x.x.x), Copyright (C) 200x Mathematical Systems
Inc.
Model description: knapsack.smp
knapsack.cc
knapsack.cc
```

この操作でこの数理計画モデルに対応するクラス定義と実装が genClass を起動したフォルダに作成されます．

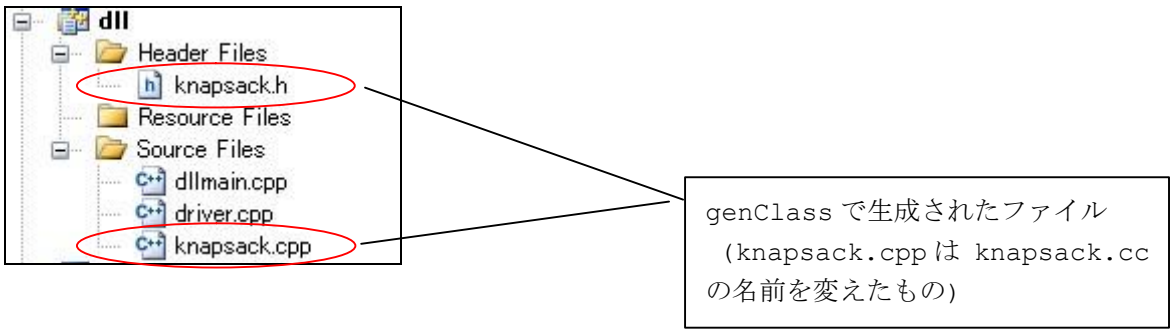
knapsack.h	knapsack.smp に対応するクラスの定義
knapsack.cc	同クラスの実装
knapsackControl.cc	利用方法サンプル

クラスの定義（knapsack.h）は，この数理計画問題に対して操作を行うコントロール手続き（C++のコード）の中で操作を行う前に必ずインクルードする必要があります．クラスの実装（knapsack.cc）の内容は通常ユーザが意識する必要はありませんが，実行形式を作成する際にユーザのプログラムにコンパイル・リンクする必要があります．
ソリューション nuoptvcapp のプロジェクト

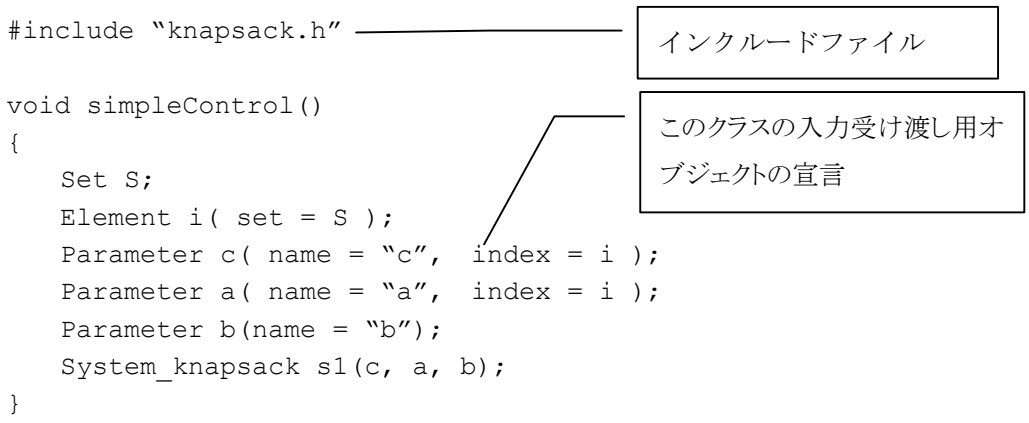
loadmodule

がこのインタフェースを用いてナップサック問題を解くプログラム例ですが，genClass で生成された実装（knapsack.cc）とクラスの定義(knapsack.h)を追加しています．ここで一点注意があります．

生成された knapsack.cc はプロジェクトに追加する前に knapsack.cpp と拡張子を変更してください．こうすることにより，VC++の GUI の規約により，このファイルが C++のコードであると認識されるようになります．



knapsackControl.cc は knapsack.smp で定義したクラスの利用方法のサンプルが記載されています. このコードは一度最適化実行を行うというものです. 入力オブジェクトがあるクラスの場合には受け渡し用オブジェクトの宣言がかかれていますので, クラスの利用のコード作成時には参考になります.



3.1.3 ドライバ VC++

ではこのモデルを解く汎用の C++プログラムを準備しましょう. それはプロジェクトに追加されている

driver.cpp

というファイルで, knapsackSolve という名前の手続きです. このルーチンのインタフェースは入力として, knapsack 問題のモデルの

Parameter a(index=i);

Parameter b;

Parameter c(index=i);

に相当するデータ（aarg,barg,carg）を与えると

Variable x(index=i);

に相当する変数値（xarg）と目的関数（farg）を返すというものです. ドライバのこのインタフェースは問題に特化して調整できるので, より柔軟なコード作成が可能になります.

```
#include "knapsack.h"   // インクルード宣言
//
//  ナップサック問題の求解
//
//
int knapsackSolve(int narg // 問題のサイズ
                ,double* aarg  // 係数 a
                ,double barg   // 係数 b
                ,double* carg  // 係数 c
```

```
                ,double* xarg  // 解ベクトル( x )
                ,double* farg  // 目的関数
                )
{
  // 初期化
  SimpleInitialize();
  { // SimpleInitialize() のコールの後は { を付ける
    // 値転送用のオブジェクト
    Set S;
    Element i(set=S);
    Parameter b;
    Parameter c(index=i),a(index=i);

    // C 配列を初期化用のデータに与える.
    b = barg; // スカラはそのまま代入
    c.readD(narg,carg); // 配列から SIMPLE のオブジェクトへの設定
    a.readD(narg,aarg); // 配列から SIMPLE のオブジェクトへの設定

    //
    // knapsack 問題の求解
    //
    System_knapsack knap(c,a,b);

    int len;
    int* ind;
    double* knapx;
    knap.x.val.dump(len,ind,knapx);
    // knapsack 問題の x を C の配列である knapx に設定

    // 解を戻り配列に設定
    int it;
    for ( it = 0 ; it < len ; ++it ) {
      xarg[it] = knapx[it];
    }
    *farg = result.optValue;
    // 目的関数値を取る簡単な方法( asDouble() でも可能 )

    // 不要な領域の破壊
    delete [] ind;
    delete [] knapx;
  } // SimpleClearBuffer() のコールの前を } で閉じる.
       // 終了処理
       SimpleClearBuffer();
       return result.errorCode; // エラーコードを返す.
}
```

SimpleInitialize()と SimpleClearBuffer()は SIMPLE を利用した処理の最初と最後にかならず必要なコールです．実装上の理由により，SimpleInitialize のコールより後と，SimpleClearBuffer()のコールより前は { } でくくる必要があります．SIMPLE のオブジェクト（Set や Element）の宣言は，この { } の中で行います．SimpleInitialize をコ

ールせずに SIMPLE のオブジェクトを宣言して利用したり，この {} の外で SIMPLE のオブジェクトを宣言すると実行時エラーとなります.

手続きの中ほどで System_knapsack というクラスのオブジェクト knap を宣言していますが，これが knapsack.smp というモデル記述に対応するクラスのオブジェクトの宣言です．このプログラムの先頭の

```
#include “knapsack.h”
```

がこのモデル定義を含むファイルで，モデル knapsack に対応するクラスを利用する場合には必ずインクルードする必要があります.

一般に NAME.smp なるモデルには System_NAME というクラスが対応します．クラスの定義ファイル NAME.h には System_NAME の定義が書かれています．そのため，System_NAME を使用する際には必ず NAME.h のインクルードが必要になります.

クラス宣言の中で SIMPLE オブジェクトを required というキーワード付きで

```
// knapsack.smp の中
Parameter c(index=i,required);
Parameter a(index=i,required);
Parameter b(required);
```

のように宣言しているので，

呼び出し側の手続きで同様に宣言した受け渡し用のオブジェクト

```
// driver.cpp の中
Set S;
Element i(set=S);
Parameter b;
Parameter c(index=i),a(index=i);
```

に

```
b = barg; // スカラはそのまま代入
c.readD(narg,cary); // 配列は readD を用いる.
a.readD(narg,aary); // 配列は readD を用いる.
```

として値を設定，システムオブジェクト knap の宣言の際に

```
System_knapsack knap(c,a,b); // knapsack 問題の求解
```

のように渡しています（この引数にはモデル中，required というキーワード付きで宣言したオブジェクトが出現順に並びます)．

3.1.4 C++関数からの呼び出し VC++

次は knapsackSolve を C++で呼び出しているメインルーチンを記述します．ここでは，モデルの b（制約式の上限）に相当するデータを動かしてナップサック問題を繰り返し解くという手続きを作成してみます．問題としては

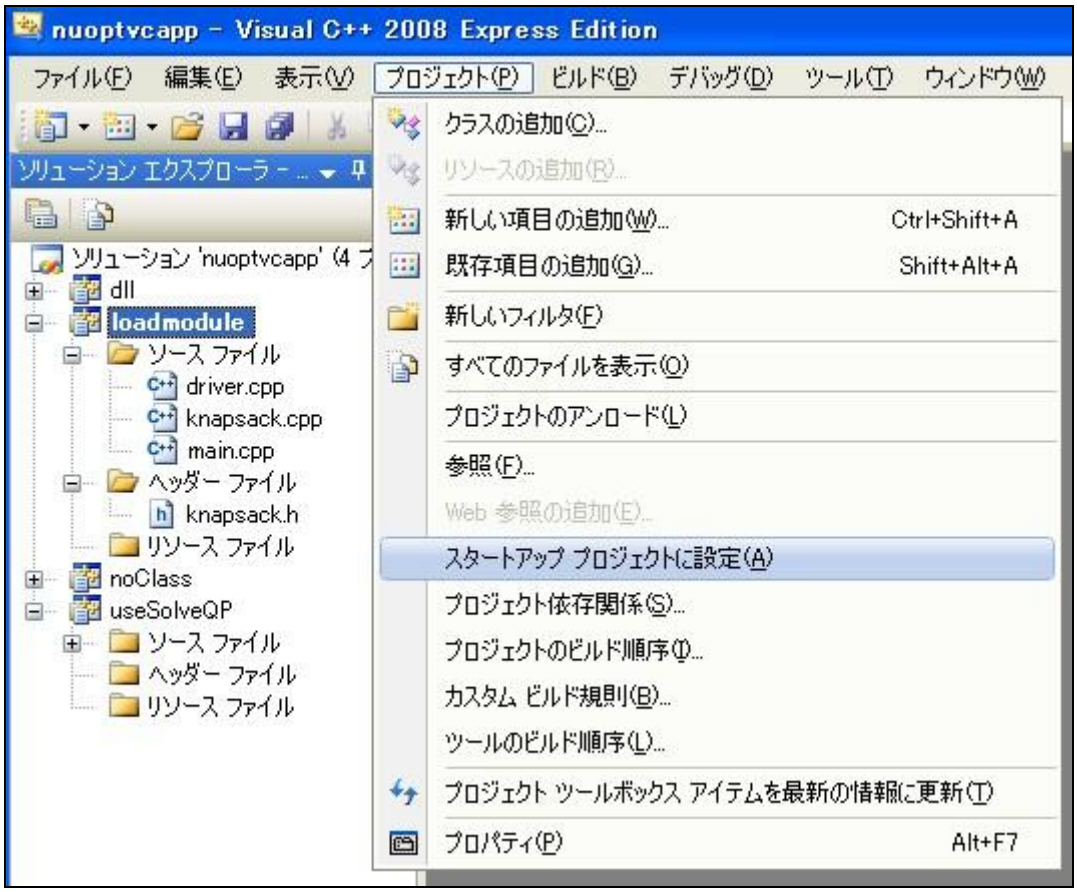
目的関数の係数：$c = (6 \quad 8 \quad 4 \quad 3 \quad 4)$
制約式の係数：　$a = (4 \quad 2 \quad 3 \quad 6 \quad 7)$

という 5 変数の簡単なものとし，制約式の上限を 1 から 30 の間で動かします．このルーチンもプロジェクト loadmodule に追加されています．

```
#include <stdio.h>

// main()の中でコールする最適化手続き（最適化ライブラリ）
int knapsackSolve(int narg,double* aarg,double barg,double* carg
                        ,double* xarg,double* farg);
void main()
{
  // 問題のデータ
  int narg = 5;
  double aarg[5] = {4,2,3,6,7};
  double carg[5] = {6,8,4,3,4};
  double xarg[5];
  double barg = 20;
  double farg;

  int errCode;

  // barg を変化させながら knapsackSolve を繰り返しコールする
  int ib;
  for ( ib = 1 ; ib <= 30; ++ib ) {
    barg = (double) ib;
    errCode = knapsackSolve(narg,aarg,barg,carg,xarg,&farg);
    printf("errCode=%d b=%g obj=%g ",errCode,barg,farg);
    printf(" x= ");
    int i;
    for ( i = 0 ; i < narg ; ++i ) {
      printf("%1.0f ",xarg[i]);
    }
    printf("¥n");
  }
```

}

このアプリケーションを前項のライブラリ部分(knapsackSolve()と knapsack.cc と NUOPT のライブラリを接続したもの）とリンクするとロードモジュールが作成できます.
VC++のメニューの「プロジェクト」から「スタートアップ プロジェクトに設定」を選び，次のようにこのプロジェクトを選択します.



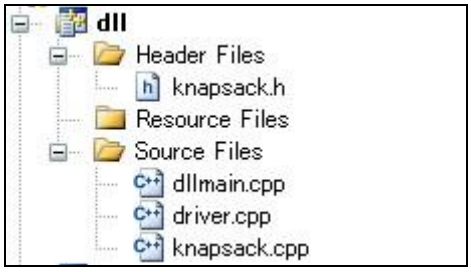
つづいて VC++GUI の実行を選択：

次のような出力が得られます.

```
errCode=0 b=1 obj=0  x= 0 0 0 0 0
errCode=0 b=2 obj=8  x= 0 1 0 0 0
errCode=0 b=3 obj=8  x= 0 1 0 0 0
errCode=0 b=4 obj=8  x= 0 1 0 0 0
errCode=0 b=5 obj=12  x= 0 1 1 0 0
errCode=0 b=6 obj=14  x= 1 1 0 0 0
        (中略)
errCode=0 b=26 obj=25  x= 1 1 1 1 1
errCode=0 b=27 obj=25  x= 1 1 1 1 1
errCode=0 b=28 obj=25  x= 1 1 1 1 1
errCode=0 b=29 obj=25  x= 1 1 1 1 1
errCode=0 b=30 obj=25  x= 1 1 1 1 1
Press any key to continue
```

3.1.5 VisualBasic からの呼び出し VC++

ライブラリ部分を DLL にすると一般に VisualBasic などのアプリケーションから呼ぶことができます. プロジェクト dll は同一のドライバルーチンを使って DLL を作成するプロジェクトです.



DLL の入り口となる knapsackSolve() をコールするルーチンを作成すると以下のようになります. このルーチンは入力を knapsackSolve に渡して一度だけ解くという操作を行います.

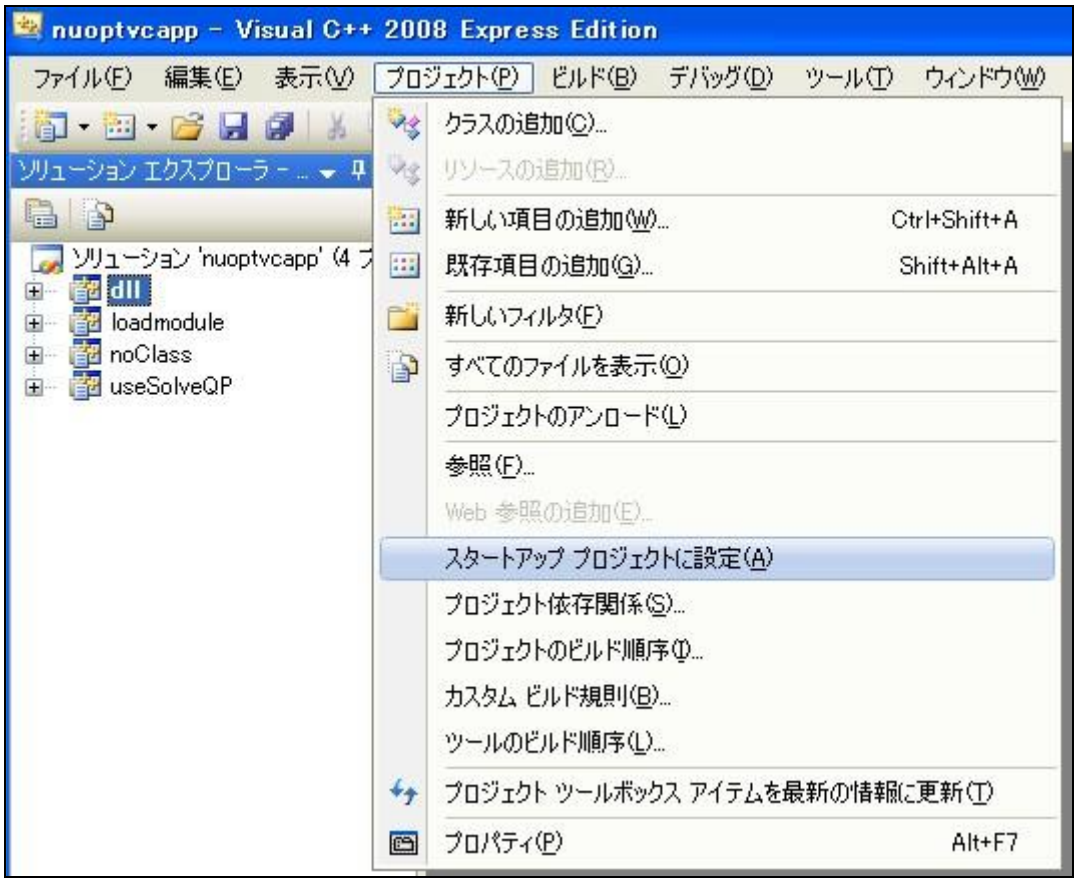
```
#include <windows.h>
#include <stdio.h>

// SIMPLE で書かれた NUOPT 側メイン
int knapsackSolve(int narg,double* aarg,double barg,double* carg
           ,double* xarg,double* farg);

// VB から呼ばれる DLL の入り口
#define EXPORT __declspec (dllexport)
#define FORVB _stdcall

extern "C" long EXPORT FORVB nuoptmain(
                                  long narg // 問題サイズ
                                  ,double* aarg // ベクトル a
                                  ,double  barg // ベクトル b
                                  ,double* carg // ベクトル c
                                  ,double* xarg // ベクトル x(出力)
                                  ,double* farg // 目的関数値 (出力)
                                  )
{
  int errCode = knapsackSolve(narg,aarg,barg,carg,xarg,farg);
}
```

DLL のデバッグは

- 出力が得られない
- 異常が起こると起動アプリごと動作を停止する

など, 困難な点が多いので一度ダミーのメイン関数でコールするロードモジュールなどで動作を確認したのち, DLL を作成することをお勧めします. DLL のメインの宣言方法は通常の Windows アプリケーションと同様です. ここでは, 生成した DLL を Excel の VBA からコールする例を示しますのでここでは VBA から呼ばれることを前提とした宣言 (_stdcall) となっています. まず, DLL を作成しましょう.

VC++のメニューの「プロジェクト」から「スタートアップ プロジェクトに設定」を選び, 次のようにこのプロジェクトを選択します.



こうして，ビルドメニューから

ビルドを選択すると，DLL が作成されます．では，これをリンクして実行する VBA 側の設定を行います．この DLL を読み込んで実行するサンプルアプリケーションは

（NUOPT のインストール場所）¥SAMPLES¥app¥dll¥dllsample.xls

です．「マクロを有効にする」で，このエクセルファイルを開き，VisualBasic エディタを開くと，Sheet1 に

は次のようなコードが書かれています. これはシートからデータを読み取って nuoptmain をコールするコードです. 下線部は DLL の絶対パスです(先ほどアクティブな構成の設定でデバッグを選択するとこの場所に DLL が作成されます). C++の int は VBA の Long 型に, double は VBA の Double 型にそれぞれ対応します.

Sheet1 に含まれるコード:

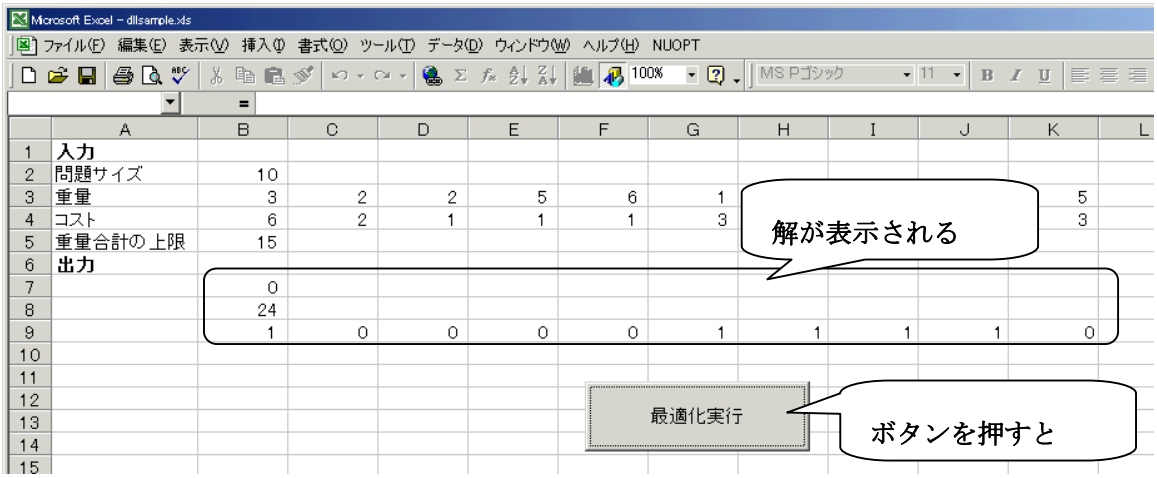
```
Private Sub execbutton_Click()
    Dim a() As Double
    Dim c() As Double
    Dim x() As Double
    Dim n As Long
    Dim obj As Double
    n = Range("B2")
    ReDim a(n)
    ReDim c(n)
    ReDim x(n)
    For i = 1 To n
        a(i - 1) = Range("A3").Offset(0, i)
        c(i - 1) = Range("A4").Offset(0, i)
    Next i
    b = Range("B5")
    code = nuoptmain(n, a(0), b, c(0), x(0), obj)
    Range("B7") = code
    Range("B8") = obj
    For i = 1 To n
        Range("A9").Offset(0, i) = x(i - 1)
    Next i
End Sub
```

実行させるには, リンクを行う必要がありますが, そのためには Module1 に次のように宣言を行います.

DLL の宣言:

```
Declare Function nuoptmain Lib
"c:¥Program Files¥nuopt¥samples¥app¥dll¥Release¥dll.dll" Alias _
"_nuoptmain@28" (ByVal n As Long, ByRef a As Double, ByVal b As
Double , ByRef c As Double, ByRef x As Double, ByRef f As Double)
As Long
```

エクセルシートに戻ってボタンを押すと DLL の実行が行われます. 以下はこのコードを使ったアプリケーションの実行例です.



3.1.6 GUI でドライバ（コントロールルーチン）や WINAPI を利用する VC++

.smp から生成されたコードを呼び出すのが C++関数である場合には次の方法で一つの.smp にまとめてしまうと，GUI での実行が可能となります7.

.smp の末尾に

```
%%%%% With_simpleControl %%%%%
```

という行を追加して，

```
void simpleControl()
{
}
```

という手続きの定義を含む C++のコードを書き，すべてを.smp ファイルだとしてセーブし，通常のモデルと同様に利用します．すると，simpleControl()が常に最初にコールされ，.smp で定義したモデルに対応するクラスをこの手続き内で使えますので，simpleControl()をドライバソースのメインとして用いることができるようになります．

また，

```
%%%%% #include "C:¥mysrc¥myheader.h"  %%%%
```

という行を挟めばモデルとコントロールで共通にこのインクルードが行われます．

この機能を使うと，例えば次のように，WINAPI をモデル定義中に用いることができます．次

[7] NUOPT の Ver.4（あるいは Ver.5 の Windows98/Me 対応版）でコントロールルーチンをカスタマイズされている場合にはこの方法によって Ver.6 の GUI に置き換えることができます．置き換えの過程にて問題が生じましたら nuopt-support@msi.co.jp にご連絡ください．

のモデルはWINAPI（getdate）を用いて，時刻を表示するものです.

```
//
// 簡単な LP （最適化の実行日付を表示する）
//
Variable x;
Variable y;
// 目的関数
Objective cost(type=minimize);
cost = 180*x + 160*y;
// 制約
6*x + y >= 12;
3*x + y >= 8;
4*x + 6*y >= 24;
// 変数の上下限
0 <= x <= 5;
0 <= y <= 5;

char* getdate(char*);

char chrTime[128];
printf("%s¥n",getdate(chrTime));
fflush(stdout);

%%%%% #include <time.h> %%%%%

// C++ の関数の定義
char* getdate(char* chrTime)
{
       _strdate(chrTime); // mm/dd/yy 形式で日付を文字列に格納
       return chrTime;
}
```

3.2 BC++/UNIX 版ライブラリ (SIMPLE モデル記述からクラスを生成して利用する例)

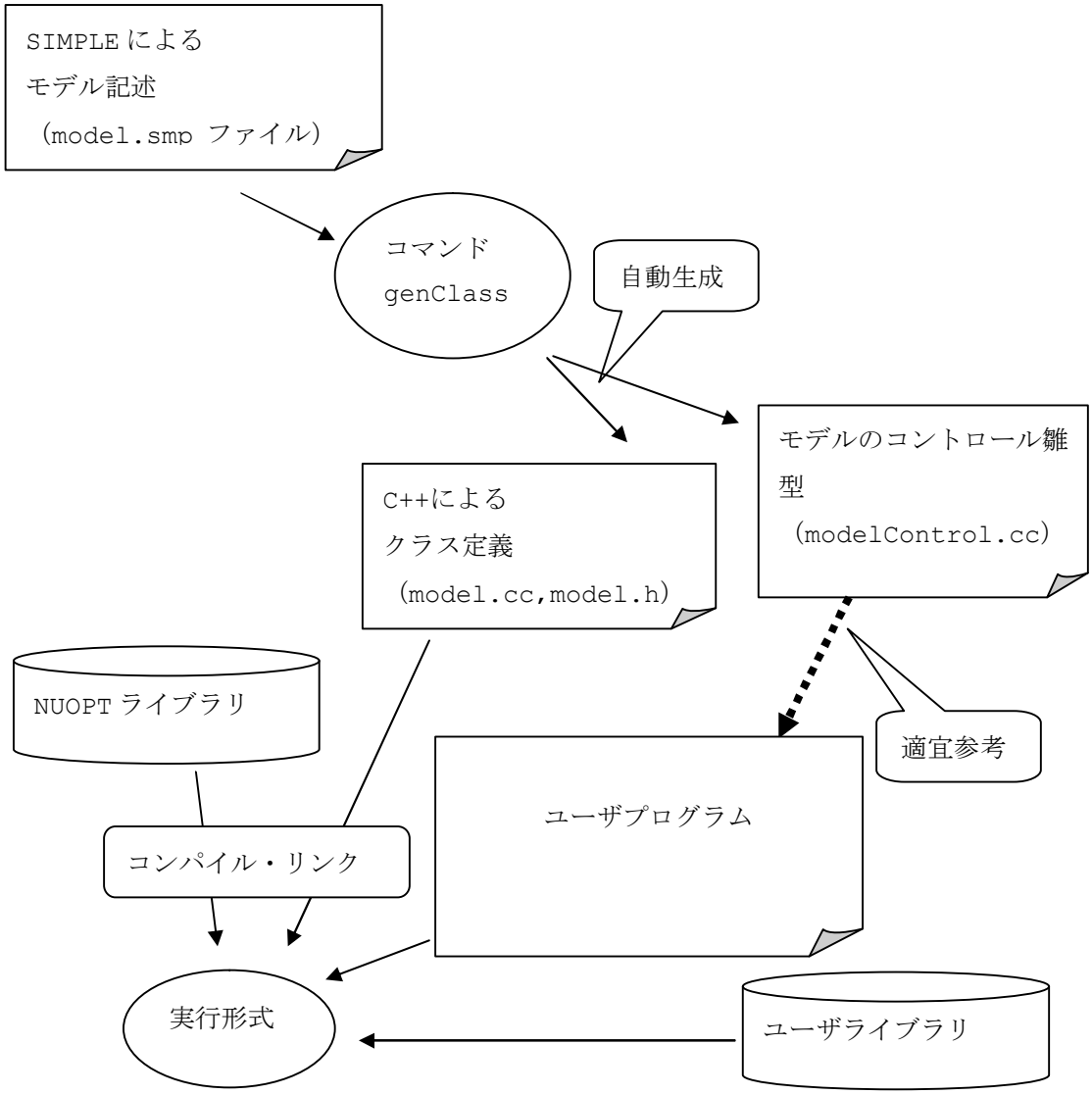
3.2.1 手順の概要 BC++-UNIX

次のページに関連するファイルの関係を図示します. ユーザはまず, モデリング言語 SIMPLE を用いてモデル記述を行います. モデルが記述できたら, その内容を.smp という拡張子のファイル（例えば model.smp）に格納します.

次に genClass というコマンドを用いて model.smp から対応するクラス宣言（model.h）とクラスの実装（model.cc）を自動生成します. ユーザはこの内容を意識する必要はありません.

こうして生成されたクラス宣言（model.h）をインクルードすることによって, ユーザのプログラムからこのモデルに対してデータを入力したり, 求解を指示したり（solve()）, 解を出力したり(cout<<,print(),dump())する操作が可能になります.

実行形式の生成の際には, 先ほど自動生成されたクラス実装をコンパイル, リンクします. こうして生成された SIMPLE のクラスは同時に何個でも,また何回でも利用することができます. ただし SIMPLE のクラスを利用するに先立って, ユーザのプログラムから初期化関数を呼び出す必要があります（詳細は後述します）.


SIMPLE による
モデル記述
（model.smp ファイル）
コマンド
genClass
自動生成
モデルのコントロール雛型
（modelControl.cc）
C++による
クラス定義
（model.cc,model.h）
NUOPT ライブラリ
適宜参考
ユーザプログラム
コンパイル・リンク
実行形式
ユーザライブラリ

3.2.2 二次計画問題の例 BC++･UNIX

サンプルディレクトリの中には次のような簡単な二次計画問題を記述したモデルファイル QP.smp があります.

1. モデルの記述 BC++･UNIX

```
//
//  SIMPLE チュートリアル
//  3.1.2 ２次計画問題
//  より
    // 集合
    Set S;
    Element i(set=S);
    Set T;
    Element j(set=T), k(set=T);

    // パラメータ
    Parameter c(name="c", index=j);
    Parameter Q(name="Q", index=(j,k));
    Parameter cu(name="cu", index=i);
    Parameter cl(name="cl", index=i);
    Parameter A(name="A", index=(i,j));
    Parameter bu(name="bu", index=j);
    Parameter bl(name="bl", index=j);

    // 変数
    Variable x(index=j);

    // 最小化
    Objective f(type=minimize);
    f = sum(c[j] * x[j], j)
+ 1.0/2.0 *sum(Q[j,k]*x[j]*x[k], (j,k));

    // 条件
    Expression constr(index=i);
    constr[i] = sum(A[i,j] * x[j], j);
    cu[i] >= constr[i] >= cl[i];
    bu[j] >= x[j] >= bl[j];
```

この問題は，マニュアルセット中の「SIMPLE チュートリアル」の「３．１．２．二次計画問題」で解説されている例で，一般的な二次計画問題を記述するモデルです．モデルの記述についての解説は「SIMPLE チュートリアル」をご参照ください.

2. genClass の起動 BC++・UNIX

さて,次はこのモデル記述 QP.smp からクラス定義と実装を記述したファイル(C++のコード)を生成します．そのためには Windows 環境ともにプロンプトから

```
> genClass QP.smp
```

とします．genClass は NUOPT をインストールした環境でのみ実行可能なユーティリティーです．そうすると，この数理計画モデルに対応するクラス定義と実装が genClass を起動したディレクトリに作成されます．サンプルの makefile ではこの操作を生成規則に記述しています．

QP.h	QP.smp に対応するクラスの定義
QP.cc	同クラスの実装
QPControl.cc	コントロール手続きサンプル

クラスの定義（QP.h）は，この数理計画問題に対して操作を行うコントロール手続き（C++のコード）の中で操作を行う前に必ずインクルードする必要があります．クラスの実装（QP.cc）の内容は通常ユーザが意識する必要はありませんが,実行形式を作成する際にユーザのプログラムにコンパイル・リンクする必要があります.

3. モデルからの問題の生成, 求解, 出力 BC++・UNIX

次にこの数理計画モデルに対する操作の例です．以下はサンプルプログラム useClass.cc からこの二次計画問題に関連する操作を行っている部分を抜き出したものです.

```
#include "QP.h"

   …中略…

void simpleControl()
{
  int len,i,*ind;

  //
  //  QP の求解
  //
  System_QP qp;

  qp.f.val.print();
  qp.x.val.print();
  qp.constr.val.print();

  double* x;
```

クラスの定義のインクルード（必須）

QP.smpに対するクラスオブジェクト qp の宣言と最適化実行

QP.smp 中のオブジェクトの表示

```
qp.x.val.dump(len,ind,x); // QP の x を C++ の配列にダンプ.

printf("x(QP):¥n");
for ( i = 0 ; i < len ; ++i ) {
    printf("[%3d] %10.3e ",ind[i],x[i]);
    if ( (i+1) % 4 == 0 ) {
       printf("¥n");
    }
}
printf("¥n");

   …後略…
```

useClass.cc では simpleControl という手続きの中で二次計画問題に対する操作が記述されています．手続きの名前は任意です.

手続きの最初で System_QP というクラスのオブジェクト qp を宣言していますが，これが QP.smp というモデル記述に対応するクラスのオブジェクトの宣言です.
一般に

NAME.smp なるモデルには System_NAME というクラスが対応

します．クラスの定義ファイル NAME.h には System_NAME の定義が書かれています．そのため，System_NAME を使用する際には必ず NAME.h のインクルードが必要になります.

さて，宣言して作成された qp がその数理計画モデル（問題）そのものに対応します．以下ではこれを説明のため「システムオブジェクト」と呼びます．システムオブジェクトに対して

```
qp.show();
```

とすると，問題の中身が表示されます（showSystem() と同じ）．次にシステムオブジェクトと数理計画モデルの構成要素についてですが，一般的な原則として以下があります.

System_NAME のオブジェクト s について
　　s.x は NAME.smp 中のオブジェクト x に対応する.

さらに，s.x として参照されたオブジェクトに対する

代入	=
表示	print(),cout,dump()
配列の設定	readD()

などの操作は通常の SIMPLE オブジェクトに対するのと全く同様に可能です.

例えば

```
qp.f.print();
qp.x.print();
qp.constr.print();
```

とすると，それぞれ QP.smp のモデル中の f,x,constr の値が表示されます.

デフォルト動作では宣言を行ったのみで自動的に最適化の実行が行われます. オプションの設定（options.noDefaultSolve = 1）によって，自動的に最適化の実行が行われるのを抑制することが可能ですが，その場合には

```
qp.solve();
```

として陽に求解を指示します.

サンプルルーチン useClass.cc では，QP.smp のデータ（Parameter）は与えていませんが，その場合には，SIMPLE のデータファイルの内容が自動的に設定されます．ただし，そうするにはメイン関数によってデータファイルを読みこんでバッファに蓄えておく必要があります. その方法については後述します.

求解のあと，

```
qp.x.val.dump(len,ind,x); // QP の x を C++ の配列にダンプ.
```

と行うことによって，解を C++の配列に解を書き出しています．SIMPLE では一般に

```
(SIMPLE オブジェクト).val.dump
```

によってオブジェクトを配列に書き出すことができますそれを用いた記法です.

4. モデルへのデータ入力 BC++･UNIX

モデルにデータを設定する方法は

A) SIMPLE のデータファイルを用いる.
B) C++の配列を直接設定する.

という二つがあります.

5. データファイルの利用 BC++･UNIX

useClass.cc では，QP.smp のデータ（Parameter）の設定は A)の方法で行うことを前提

としています. この場合には, このこのデータからシステムオブジェクトを作成する前にメイン関数によってデータファイルを読みこんでバッファに蓄えておく必要があります.

データを読み込むには

```
readData(FILE* fp, char* filename);
```

なる手続きをコールします. fp は C の stdio.h で宣言されているファイル構造体で, 読み込むデータファイルに対応します. データが読み込まれると, データの内容が共用バッファにプールされます. データ読み込みは繰り返すことができます. UseClass.cc の main() 手続きの中の次のコードは, 呼び出し引数として与えられたファイルをすべて読み込んで共用バッファに蓄えます.

```
  //
  // SIMPLE 形式で記述されたデータを読む
  //
  int i;
  for (i = 1 ; i  < argc; i++) {
    char *file = argv[i];
    FILE* fp;
    if (strcmp(file, "stdin") == 0) fp = stdin;
    else fp = fopen(file, "r");
    if (!fp) {
      fprintf(stderr, "can not open %s¥n", file);
      exit(1);
    }
    readData(fp, file);
  }
```

useClass.cc は QP.smp に対応する例えば次のようなデータ :

```
c = [1] -3 [2] 1;
Q =
[1,1] 11 [1,2] 0
[2,1] 0  [2,2] 22
;
cu = [1] 10 [2] 10 [3] 10;
cl = [1] -1 [2] -2 [3] 2;
A =
[1,1] -1   [1,2] 0.1
[2,1] -0.2 [2,2] -1
[3,1] 2    [3,2] 1
;
bu = [1] 1 [2] 2;
bl = [1] 0 [2] 0;
```

を読み込みますが, この時点で名前と値の対応がバッファに蓄えられます. 続いて,

```
System_QP qp;
```

が実行された段階で，QP.smp の中の Parameter である a,Q,cu,cl,A,bu,bl にこの値が設定されます. バッファにプールされたデータとパラメータの対応付けは名前によって行われます．名前はコンストラクト時に name = で与えますが，name = によって名前を与えられていないオブジェクトには，そのオブジェクトの名前と同じ名前が自動的に与えられます.

例えば，通常

```
Parameter a(name="パラメータ");
```

に対しては

```
パラメータ = 2;
```

というデータファイルの記述が対応しますが,

```
Parameter a;
```

と書くと，これは

```
Parameter a(name="a");
```

としたことに相当して，データファイル中の

```
a = 2;
```

という記述に対応します.

データファイルから読み込んだデータは大域的に有効で, 特定のモデルに結びついたものではないことにご注意ください.

例えば

```
Parameter a;
```

という記述を含む二つのモデル M1.smp,M2.smp があり, それから生成されたクラスの両方を利用するコード,

```
System_M1 m1;
System_M2 m2;
```

があるとします.
最初に

```
a = 2;
```

というデータファイルを読み込んだ状態でこのコードを実行すると，m1,m2 両方に a = 2 が設定されます.

3.2.3 ナップサック問題のモデル（C++の配列からのデータ設定）BC++･UNIX

useClass.cc ではもうひとつ knapsack.smp というモデルを利用していますが，このモデルのデータは C++の配列からデータを入力しています．その場合にはモデルの記述に変更が必要です. 具体的には外部入力とするパラメータの宣言部分に required というキーワードを追加します.

```
//
// ナップサック問題
//
//
// NUOPT/SIMPLE 使用手引より
//
//   4．ナップサック問題
//

    Set S;
    Element i(set=S);
    IntegerVariable x(index=i,type=binary); // 整数変数
    Parameter c(index=i,required);
    Parameter a(index=i,required);
    Parameter b(required);
    Objective obj(type=maximize);

    obj = sum(c[i]*x[i],i);    // 目的関数
    sum(a[i]*x[i],i) <= b;     // 制約条件
```

次は useClass.cc の中で knapsack.smp に対するデータを設定している部分です. required をつけて宣言したオブジェクトを含む .smp に対応するクラス（ここでは System_knapsack ）は，required をつけて宣言した Parameter を順に与えて宣言する必要があります．そうして与えられた Parameter が対応するオブジェクトの初期値になります.

```
// knapsack.smp の初期化用のデータの宣言
Set S;
Parameter a(index=S);
Parameter c(index=S);
Parameter b;

// データを C++ 側から与える.
double cary[10] = { 42, 12, 45 , 5  , 2, 61, 89 , 32 , 47, 18};
double aary[10] = { 39, 13, 68 , 15 , 10 , 20 , 31 , 15 , 41 , 16};
```

```
// C++ の配列を初期化用のデータに与える.
b = 121; // スカラはそのまま代入
c.readD(10,cary); // 配列は readD を用いる.
a.readD(10,aary); // 配列は readD を用いる.

System_knapsack knap(c,a,b); // knapsack 問題の求解

double* knapx;
knap.x.val.dump(len,ind,knapx); // knapsack 問題の x を knapx にダンプ
// 表示
printf("x(knapsack):¥n");
for ( i = 0 ; i < len ; ++i ) {
    printf("[%3d] %10.3e ",ind[i],knapx[i]);
    if ( (i+1) % 4 == 0 ) {
printf("¥n");
    }
}
printf("¥n");

delete [] ind;
```

上の例では，まず，useClass.cc の中で c,a,b を宣言して，readD や代入によってデータの内容を設定，続いて knap の宣言に与えています．こうすることによってモデル中の c,a,b に useClass.cc の中の c,a,b の値が与えられます． useClass.cc のなかの c,a,b と knapsack.smp の中の c,a,b とはここでは同じ名前ですが，必ずしもそうである必要はありません．

3.2.4 初期設定と main 関数 BC++･UNIX

前項のように SIMPLE のクラスをユーザプログラムから利用するには初期設定をおこなっておく必要があります.

次が simpleControl を呼び出している useClass.cc のメイン部分です.

```
#include "simple.h"
void simpleControl();

//
//  メイン関数サンプル
//
//
int main(int argc,char** argv)
{
  // SIMPLE 内部の初期化(これはいつでも必須)
  SimpleInitialize();

  //
  // SIMPLE 形式で記述されたデータを読む
//      (ファイル名が引数から与えられるものとする)
  //
  int i;
  for (i = 1 ; i  < argc; i++) {
    char *file = argv[i];
    FILE* fp;
    if (strcmp(file, "stdin") == 0) fp = stdin;
    else fp = fopen(file, "r");
    if (!fp) {
      fprintf(stderr, "can not open %s¥n", file);
      exit(1);
    }
    readData(fp, file);
  }

  simpleControl();

  //
  // SIMPLE が利用するスタティックバッファのクリア
  //
  SimpleClearBuffer();

  return 0;
}
```

必須な初期設定

モデルの求解など

このプログラムはデモ用に最低限の機能を果たすサンプルです. ユーザはこの内容にこだわることなく，自由に記述することができますが，以下に述べる注意を守る必要があります.
まず，メイン手続きで（すべての NUOPT 関連の処理を行う前に）

```
SimpleInitialize();
```

を必ず一度だけ呼ぶ必要があります．これを呼ぶためには，

```
#include "simple.h"
```

というヘッダファイルのインクルードが必要です．

実行形式を作成するには

使用するモデルファイルから生成されたクラスの実装（QP.cc に相当するもの）
NUOPT のライブラリ

をユーザプログラムにリンクする必要があります．
最後の SimpleClearBuffer()は SIMPLE が保持しているスタティックバッファのクリアを行う命令です．このコールを行った後は，コールを行う前に定義したいかなる SIMPLE オブジェクトの内容も参照することはできなくなる代わりに，プロセスの占有メモリ領域を減らすことができます．

次はメイクファイルの内容（一部）です．ライブラリはこのメイクファイルで”LIB”という変数に定義されています．

UseClass.cc では，QP.smp の他，knapsack.smp というモデルを使用しているので，これらをまず，genClass に入力，生成された QP.cc，knapsack.cc を useClass.cc とリンクしています．

```
QP.cc : QP.smp
                        genClass $*.smp

knapsack.cc : knapsack.smp
                        genClass $*.smp

useClass.exe : QP.obj knapsack.obj useClass.obj
    link -out:$@ QP.obj knapsack.obj useClass.obj $(LIBS)
```

*.smp からソースを生成する規則

実行形式作成

3.2.5 実行形式の作成と最適化の実行 BC++・UNIX

さて，サンプルコード useClass のロードモジュールを作成して実行してみましょう.

```
prompt% make useClass.exe
```

とすると, QP.cc,knapsack.cc,useClass.cc がコンパイル, リンクされて useClass.exe が作成されます. 引数として qp312.dat （QP.smp 用のサンプルデータ）を与えてロードモジュールを実行すると，以下のような出力が得られます.

```
%prompt useClass.exe qp312.dat
<reading data_file: qp312.dat>
Eqnum = 1  profileStart...展開中 目的関数 (1/3 name="f")
 profileEnd, time =     -0.000
Eqnum = 2  profileStart...展開中 制約式   (2/3)
 profileEnd, time =     -0.000
Eqnum = 3  profileStart...展開中 制約式   (3/3)
 profileEnd, time =     -0.000
NUOPT x.x.x, Copyright (C) 1991-200x Mathematical Systems Inc.
PROBLEM_NAME                                          QP
NUMBER_OF_VARIABLES                                    2
NUMBER_OF_FUNCTIONS                                    4
PROBLEM_TYPE                                  MINIMIZATION
METHOD                                       TRUST_REGION
<preprocess begin>.........<preprocess end>
<iteration begin>
   res=5.6e+00 .... 1.0e-03 . 3.3e-09
<iteration end>
STATUS                                           OPTIMAL
VALUE_OF_OBJECTIVE                            2.318182061
ITERATION_COUNT                                        7
FUNC_EVAL_COUNT                                       12
FACTORIZATION_COUNT                                   15
RESIDUAL                                 3.310027094e-09
ELAPSED_TIME(sec.)                                  0.01
SOLUTION_FILE                                 solver.sol
f=2.31818
x[1]=0.939394
x[2]=0.121213
constr[1]=-0.927272
constr[2]=-0.309091
constr[3]=2
x[  1] =  9.394e-01 x[  2] =  1.212e-01
x(QP):
```

```
[  1]  9.394e-01 [  2]  1.212e-01
** QP is solved **
Eqnum = 4  profileStart...展開中 目的関数 (1/2 name="obj")
 profileEnd, time =    -0.000
Eqnum = 5  profileStart...展開中 制約式   (2/2)
 profileEnd, time =    -0.000
NUOPT x.x.x, Copyright (C) 1991-200x Mathematical Systems Inc.
PROBLEM_NAME                                    knapsack
NUMBER_OF_VARIABLES                                   10
NUMBER_OF_FUNCTIONS                                    2
PROBLEM_TYPE                                MAXIMIZATION
METHOD                                         SIMPLEX
<preprocess begin>..........<preprocess end>
<iteration begin>
       1.2B
    up:     244.71 lo:     -1e+50               time:  0.0s:mem(Mb)=0
                                                llen:2 #prob:4 #piv:9
#1   up:        244.71  lo:            242  gap:        2.7073  time:
0.0s:mem(Mb)=0
                                                llen:2 #prob:5 #piv:10

<iteration end>
STATUS                                           OPTIMAL
VALUE_OF_OBJECTIVE                                    242
SIMPLEX_PIVOT_COUNT                                     7
PARTIAL_PROBLEM_COUNT                                  10
DUAL_SIMPLEX_PIVOT_COUNT                                4
ELAPSED_TIME(sec.)                                   0.00
SOLUTION_FILE                                  solver.sol
x(knapsack):
[  1]  1.000e+00 [  2]  0.000e+00 [  3]  0.000e+00 [  4]  0.000e+00
[  5]  0.000e+00 [  6]  1.000e+00 [  7]  1.000e+00 [  8]  1.000e+00
[  9]  0.000e+00 [ 10]  1.000e+00
** knapsack is solved **
```

上に挙げた内容は NUOPT からの求解に関するメッセージです.

```
System_QP qp;
       ….
System_knapsack knap(c,a,b);
```

として生成された問題について順に求解が行われていることがわかります. 問題の変数の数（NUMBER_OF_VARIABLES）や目的関数の値（VALUE_OF_OBJECTVE）を知ることができます. NUOPT の標準出力の内容については NUOPT/SIMPLE マニュアルをご覧ください.

3.2.6 その他の操作 BC++･UNIX

これで簡単な操作について一通り解説しましたが，例えば

- NUOPT のパラメータを指定する.
- 最適化の初期値を設定する.
- 解をいろいろなフォーマットで印刷，ファイル出力する.

などが通常の SIMPLE の記述と同様に可能です.以下ではこれらについて具体的に解説します.
NUOPT のパラメータを設定するには，通常の .smp の記法と同様に

```
options.パラメータ名 = …;
```

というコードを挿入します．例えば NUOPT の標準出力を抑制するには

```
options.outputMode = “silent”;
```

とします．また，デフォルト出力される NUOPT の求解レポート *.sol の出力を抑制するには

```
options.outfilename = “_NULL_”;
```

と設定します．例えば，useClass.cc で次のように書くと，

```
void simpleControl()
{
  // NUOPT の標準出力を抑制,
  options.outputMode = "silent";
  // NUOPT の解ファイルの出力を抑制
  options.outfilename = "_NULL_";

  int len;
  int* ind;
  int i;
     …
}
```

次のように NUOPT からの標準出力が抑制されます.

```
<reading data_file: qp312.dat>
f=2.31818
x[1]=0.939393
x[2]=0.121214
constr[1]=-0.927272
constr[2]=-0.309092
constr[3]=2
x(QP):
[  1] 9.394e-001 [  2] 1.212e-001
** QP is solved **
x(knapsack):
[  1] 1.000e+000 [  2] 0.000e+000 [  3] 0.000e+000 [  4] 0.000e+000
[  5] 0.000e+000 [  6] 1.000e+000 [  7] 1.000e+000 [  8] 1.000e+000
[  9] 0.000e+000 [ 10] 1.000e+000
** knapsack is solved **
```

変数の初期値の設定はシステムオブジェクト内の変数に相当するオブジェクトに対して代入を行います. すなわち useClass.cc の例では

```
System_QP qp;
qp.x[1] = 2.0; // x[1]の初期値の設定
```

のようにして行います. ただし, デフォルト動作では qp を宣言した段階で, 初期値の設定を行う前に求解が行われてしまいますので, これを抑制し, 陽に求解を指示するようにしましょう. 次のようにします.

```
options.noDefaultSolve = 1; // デフォルトで求解を行わないようにする
System_QP qp;      // qp の宣言（求解は行わない）
qp.x[1] = 2.0;     // 初期値の設定
qp.solve();         // 求解を指示
```

変数のすべてのコンポーネントに同じ初期値を設定するなどの場合には, モデルの中の集合や要素オブジェクトを利用して

```
qp.x[qp.j] = 2.0;  // モデル中で x[j] = 2.0; とするのと同じ
```

あるいは

```
Element j(set=qp.T); // モデル内の T をまわる要素を宣言する.
qp.x[j] = 2.0;         // j の回る範囲について  x の初期値を設定する.
```

などの SIMPLE 独特の記述方法が可能です．ここで，通常の　int 型のループ変数を使って

```
int j;
for ( j = 1 ; j <= 3 ; ++j ) {
        qp.x[j] = 2.0;
}
```

のように書くこともできますが，特に大規模なモデルの場合（ループ長が１０００以上になるような場合），には，Element をつかった表現にすることをお勧めします．これは SIMPLE の処理は Element の利用を前提として最適化されているためです．

simple_printf() を利用すると，オブジェクトの内容を書式を設定して出力することができます．例えば　useClass.cc で求解を行ったのちに，

```
simple_printf("x[%3d] = %10.3e ",qp.j,qp.x[qp.j]);
```

と書くと

```
x[  1] = 9.394e-001 x[  2] = 1.212e-001
```

のような出力が得られます（同様の書式で simple_fprintf(fp,…);とすれば任意のファイルに結果を出力することができます）．

4．SIMPLE モデルとドライバを分離しない例

本項でも同様にモデリング言語を使った問題定義から同じくナップサック問題を解くライブラリを作成しますが, ここではモデルそのものとモデルに対しての操作を行う手続き（ドライバ）を区別せずに一緒に書く方法を紹介します. これは前項のクラスを生成する方法と比べて簡便ですが NUOPT のライブラリがリンクされているプログラム全体で同時に一つの問題のみしか定義できないという制限があります.

➢ VC++

この例に対応するのが，ソリューション nuoptvcapp のプロジェクト

noClass

です.

➢ UNIX版

この例に対応するのが，ファイル

（NUOPT のインストール場所）/userapp/useSimple.cc

/MIP.cc

です.

VC++版ライブラリをお使いの方は 4.1VC++版ライブラリ(SIMPLE モデルとドライバを分離しない例)へ BC++/UNIX 版ライブラリをお使いの方は 4.2BC++/UNIX 版ライブラリ(SIMPLE モデルとドライバを分離しない例)へお進みください.

4.1 VC++版ライブラリ(SIMPLEモデルとドライバを分離しない例)

4.1.1 モデル兼ドライバの記述 VC++

このプロジェクトでは noClass.cpp というソースファイルにモデルとモデルを利用する手続きが両方書き込まれています. noClass.cpp では knapsackSolve という名前の手続きを定義しています. この手続きは入力として, knapsack 問題のモデルの

```
Parameter a(index=i);
Parameter b;
Parameter c(index=i);
```

に相当するデータ（aarg,barg,carg）を与えると

```
Variable x(index=i);
```

に相当する変数値（xarg）と目的関数（farg）を返します. このインタフェース仕様はクラスを利用するインタフェース（プロジェクト loadmodule, dll）に含まれていた driver.cpp というソースに含まれているルーチンと同じです.

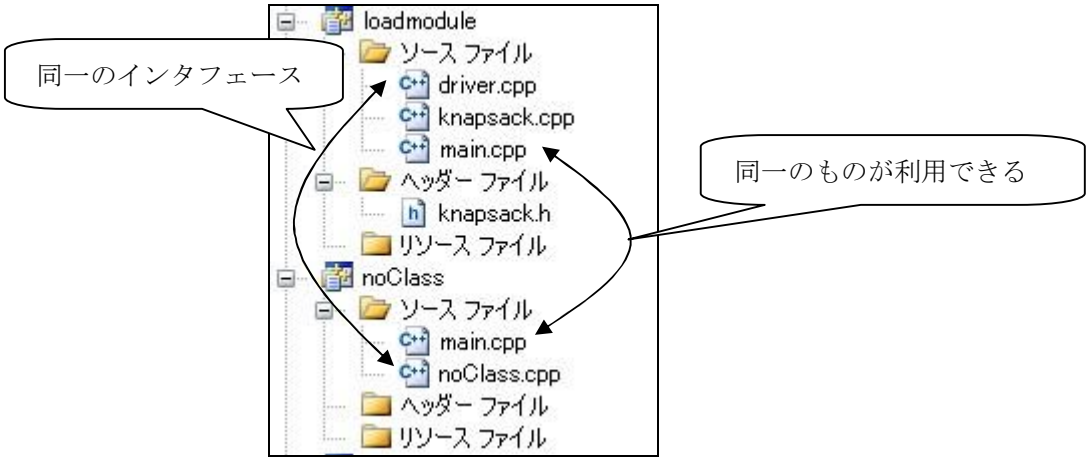
```
#include "simple.h"

//
//   ナップサック問題（モデルの記述と実行を分離しない）
//
int knapsackSolve(int narg,double* aarg,double barg,double* carg
                ,double* xarg,double* farg)
{
  //
  // システムの初期化.
  //
  SimpleInitialize();
  {
    Set S;
    Element i(set=S);

    IntegerVariable x(name="決定ベクトル"
                   ,index=i,type=binary); // 整数変数
    Parameter c(name="価値",index=i);
    Parameter a(name="重量",index=i);
    Parameter b(name="許容重量");
    Objective obj(name="総価値",type=maximize);

    // 引数からデータを設定(NUOPT/SIMPLE マニュアル第一部・5.7 節を参照)
    a.readD(narg,aarg);
    b = barg;
    c.readD(narg,carg);

    sum(a[i]*x[i],i) <= b;     // 制約条件
    obj = sum(c[i]*x[i],i);    // 目的関数
```

```
    // NUOPT からの最適化結果ファイル.sol の名前の設定
    //    options.outfilename = "nuoptout";
    // .sol のファイル出力を抑制する場合には次のように書く
    options.outfilename = "_NULL_";
    // 出力を抑制する
    options.outputMode = "silent";

    // 最適化の実行
    solve();

    // x の内容を C++ の配列にダンプ
    // (NUOPT/SIMPLE マニュアル第一部・5.6 節を参照)
    int lenx;
    int* indx;
    double* valx;
    x[i].val.dump(lenx,indx,valx);

    if ( lenx != narg) {
      return 99; // x[i] の実際の長さが narg と異なる(データに矛盾あり)
    }

    // 引数に設定
    int it;
    for ( it = 0 ; it < lenx ; ++it ) {
      xarg[indx[it]-1] = valx[it];
      // indx[it] は 1,2, ... narg なので, 1 を引く
    }

    // 目的関数値の設定
    *farg = result.optValue;

    // indx,valx は dump 内部で独自に allocate されるので free しておく.
    delete [] indx;
    delete [] valx;
  }
  // NUOPT のエラーコードを返す.
  return result.errorCode;
}
```

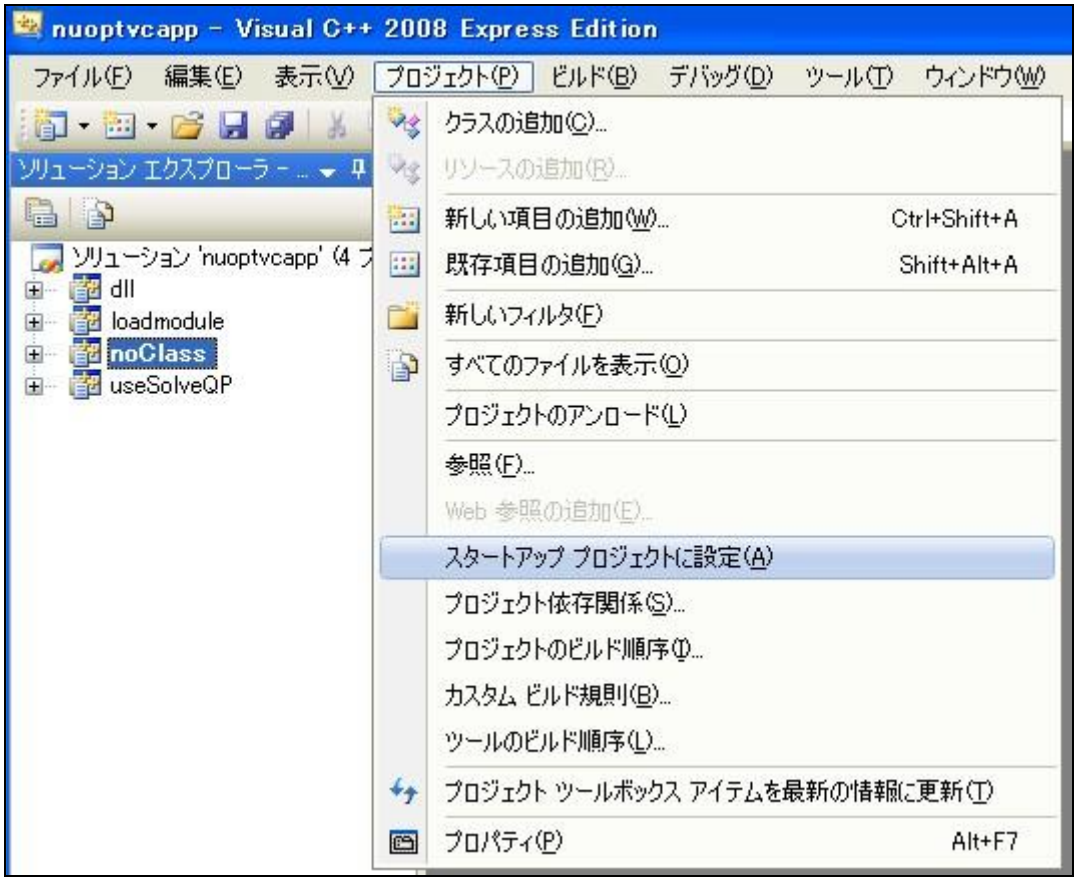
このインタフェースを用いる場合，モデルはグローバルなものを 1 つ定義することのみが可能です. 変数, 制約式は定義した場所によらずにそのグローバルなモデルに追加されてゆくことになります．SimpleInitialize()のコールより後と，SimpleClearBuffer()のコールより前は { } でくくる必要があります．SIMPLE のオブジェクト（Set や Element）の宣言は，この { } の中で行います．SimpleInitialize()をコールせずに SIMPLE のオブジェクトを宣言して利用したり，この {} の外で SIMPLE のオブジェクトを宣言すると実行時エラーとなります.

4.1.2 モデル兼ドライバのコール VC++

noClass.cc で定義された knapsackSolve は プロジェクト loadmodule, dll に含まれている driver.cpp と同一の仕様ですので，プロジェクト loadmodule のメインルーチンをそのまま使うことができます.



VC++のメニューの「プロジェクト」から「スタートアップ プロジェクトに設定」を選び，次のようにこのプロジェクトを選択します.



つづいて VC++GUI の実行を選択：

すると，プロジェクト　loadmodule　の出力と同一の出力が得られます.

4.2 BC++/UNIX 版ライブラリ (SIMPLE モデルとドライバを分離しない例)

4.2.1 モデル兼ドライバの記述 BC++･UNIX

次は，SIMPLE の記述を利用して問題を入力，求解，解の出力を行う手続き（サンプルディレクトリ内の MIP.cc）です.

```
#include "simple.h"

//
// NUOPT/SIMPLE 使用手引より
//
//   4．ナップサック問題
//
int MIPsolve(int narg,double* aarg,double barg,double* carg
           ,double* xarg,double* farg)
{
  //
  // システムの初期化.
  //
  SimpleInitialize();

    Set S;
    Element i(set=S);

    IntegerVariable x(name="決定ベクトル"
                  ,index=i,type=binary); // 整数変数
    Parameter c(name="価値",index=i);
    Parameter a(name="重量",index=i);
    Parameter b(name="許容重量");
    Objective obj(name="総価値",type=maximize);

    // 引数からデータを設定(NUOPT/SIMPLE マニュアル第一部・5.7 節を参照)
    a.readD(narg,aarg);
    b = barg;
    c.readD(narg,carg);

    0 <= x[i] <= 1;

    sum(a[i]*x[i],i) <= b;     // 制約条件
    obj = sum(c[i]*x[i],i);    // 目的関数

    // NUOPT からの最適化結果ファイル.sol の名前の設定
    //    options.outfilename = "nuoptout";
    // .sol のファイル出力を抑制する場合には次のように書く
    options.outfilename = "_NULL_";
```

引数ならびと関数名は任意

これまで定義したシステムをクリア，
以降 SIMPLE の記述

```
    // 出力を抑制する
    options.outputMode = "silent";

    // 最適化の実行
    solve();

    // x の内容を C++ の配列にダンプ
    // (NUOPT/SIMPLE マニュアル第一部・5.6 節を参照)
    int lenx;
    int* indx;
    double* valx;
    x[i].val.dump(lenx,indx,valx);

    if ( lenx != narg) {
      return 99; // x[i] の実際の長さが narg と異なる(データに矛盾あり)
    }

    // 引数に設定
    int it;
    for ( it = 0 ; it < lenx ; ++it ) {
      xarg[indx[it]-1] = valx[it];
        // indx[it] は 1,2, ... narg なので, 1 を引く
    }

    // 目的関数値の設定
    *farg = result.optValue;

    // indx,valx は dump 内部で独自に allocate されるので free しておく.
    delete [] indx;
    delete [] valx;

    // NUOPT のエラーコードを返す.
    return result.errorCode;
}
```

このインタフェースを用いる場合,モデルはグローバルなものをひとつ定義することのみが可能です. 変数, 制約式は定義した場所によらずにそのグローバルなモデルに追加されてゆくことになります.

この手続き（MIPsolve）は呼ばれるたびに，新しいデータをもとにシステムを作成し直すことを前提としておりますので手続きの始まるときグローバルなモデルの内容をクリアする必要があります．そうするにはここで行っているように最初に SimpleInitialize()をコールします.

4.2.2 モデル兼ドライバのコール BC++･UNIX

次は SIMPLE の記述を含む手続き MIPsolve をコールするメイン関数です.

```
#include <stdio.h>
#include "simple.h"

// 宣言(必須)
extern "C" {
  extern void secini();
}
// SIMPLE で書かれた最適化手続き
int MIPsolve(int narg,double* aarg,double barg,double* carg
           ,double* xarg,double* farg);

int readData(FILE* fp,char* filename); // データを読む

int main()
{
  //
  // 以下は必須( 初期化 )
  //
  secini();
  SimpleInitialize();


  int narg = 5;
  double aarg[5] = {4,2,3,6,7};
  double carg[5] = {6,8,4,3,4};
  double xarg[5];
  double barg = 20;
  double farg;

  int errCode;

  //
  // barg を変化させながら NUOPT を繰り返しコールする
  //
  int ib;
  for ( ib = 1 ; ib <= 30; ++ib ) {
    barg = (double) ib;
    //
    // システムの定義と求解.
    //
    errCode = MIPsolve(narg,aarg,barg,carg,xarg,&farg);
    printf("errCode=%d b=%g obj=%g ",errCode,barg,farg);
    printf(" x= ");
    int i;
    for ( i = 0 ; i < narg ; ++i ) {
      printf("%1.0f ",xarg[i]);
    }
    printf("¥n");
  }

  //
  // SIMPLE が保持している static バッファのクリア
  //
  SimpleClearBuffer();
```

```
  return 0;
}
```

SIMPLE を記述した手続きを呼ぶ場合には，secini() と SimpleInitialize()を呼ぶ必要があります．ここでは，データファイルを読み込まずに C++の配列にデータを設定して MIPsolve を直接呼んでいますが，以下のようにしてデータファイルをよみこんでおくこともできます．

```
char* filename = "c:¥¥temp¥¥data.dat";
FILE* fp = fopen(filename,"r");  // ファイルを開く
readData(fp, filename); // データ
fclose(fp); //
```

こうすると，データの名前と内容がバッファに蓄えられます．蓄えられたデータは SIMPLE オブジェクトで同じ名前（name）を持つオブジェクトと対応し，宣言時に読み込まれます．

例えばデータファイル内に

```
param = 2;
```

という記述があると，以降

```
Parameter a(name=”param”);
```

という宣言によって a に 2 が代入されます．同じものが複数回現れた場合には，そのすべてにデータの内容が代入されます．例えば，

```
Variable v(name=”param”);
```

という宣言があれば，v にも 2 が代入されます． 最後の SimpleClearBuffer()は SIMPLE が保持しているスタティックバッファのクリアを行う命令です．このコールを行った後は，コールを行う前に定義したいかなる SIMPLE オブジェクトの内容も参照することはできなくなる代わりに，プロセスの占有メモリ領域を減らすことができます．

4.2.3 実行形式の作成と最適化の実行 BC++･UNIX

さて，このサンプルコード useSimple.cc,Mip.cc のからロードモジュールを作成して実行してみましょう．

```
prompt% make useSimple.exe
prompt% useSimple
```

と打ちます．実行させると，

```
errCode=0 b=1 obj=0  x= 0 0 0 0 0
errCode=0 b=2 obj=8  x= 0 1 0 0 0
errCode=0 b=3 obj=8  x= 0 1 0 0 0
errCode=0 b=4 obj=8  x= 0 1 0 0 0
errCode=0 b=5 obj=12  x= 0 1 1 0 0
errCode=0 b=6 obj=14  x= 1 1 0 0 0
errCode=0 b=7 obj=14  x= 1 1 0 0 0
errCode=0 b=8 obj=14  x= 1 1 0 0 0
errCode=0 b=9 obj=18  x= 1 1 1 0 0
errCode=0 b=10 obj=18  x= 1 1 1 0 0
errCode=0 b=11 obj=18  x= 1 1 1 0 0
errCode=0 b=12 obj=18  x= 1 1 1 0 0
errCode=0 b=13 obj=18  x= 1 1 0 0 1
errCode=0 b=14 obj=18  x= 1 1 1 0 0
errCode=0 b=15 obj=21  x= 1 1 1 1 0
errCode=0 b=16 obj=22  x= 1 1 1 0 1
errCode=0 b=17 obj=22  x= 1 1 1 0 1
errCode=0 b=18 obj=22  x= 1 1 1 0 1
errCode=0 b=19 obj=22  x= 1 1 1 0 1
errCode=0 b=20 obj=22  x= 1 1 1 0 1
errCode=0 b=21 obj=22  x= 1 1 1 0 1
errCode=0 b=22 obj=25  x= 1 1 1 1 1
errCode=0 b=23 obj=25  x= 1 1 1 1 1
errCode=0 b=24 obj=25  x= 1 1 1 1 1
errCode=0 b=25 obj=25  x= 1 1 1 1 1
errCode=0 b=26 obj=25  x= 1 1 1 1 1
errCode=0 b=27 obj=25  x= 1 1 1 1 1
errCode=0 b=28 obj=25  x= 1 1 1 1 1
errCode=0 b=29 obj=25  x= 1 1 1 1 1
errCode=0 b=30 obj=25  x= 1 1 1 1 1
```

という出力が得られます. 各行は異なるデータに対応するナップサック問題の解に対応しています.

5. 外部接続時に利用される SIMPLE のツール

プロジェクト loadmodule の driver.cpp や プロジェクト noClass の noClass.cc (BC++/UNIX 版では useClass.cc や useSimple.cc)では SIMPLE のモデルを定義するだけではなく，値を設定したり解を取得したりしていますが，このような場合に必要なツールについて解説します.

5.1 モデルから作成されたオブジェクト（システムオブジェクト）の操作

driver.cpp (BC++/UNIX 版では useClass.cc)で行っているように，genClass で生成したクラス定義から

```
System_knapsack knap(c,a,b);
```

として生成されたオブジェクト（ここでは knap）がデータを与えられて作成したその数理計画モデル（この場合には knapsack 問題）そのものに対応します（以下ではこれを説明のため「システムオブジェクト」と呼びます）．ここで

```
knap.show();
```

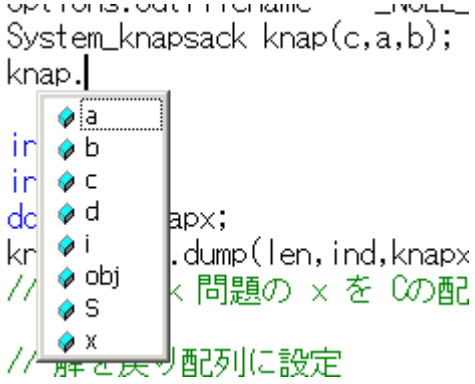
とすると，問題の中身が表示されます（showSystem() と同じ）.
システムオブジェクトと数理計画モデルの構成要素については，一般的な原則として以下があります.

```
System_NAME のオブジェクト s について
        s.x は  NAME.smp 中のオブジェクト x に対応する.
```

すなわち，

```
//
//   ナップサック問題
//
    Set S;
    Element i(set=S);
    IntegerVariable x(index=i,type=binary); // 整数変数
    Parameter c(index=i,required);
    Parameter a(index=i,required);
    Parameter b(required);
    Objective obj(type=maximize);

    obj = sum(c[i]*x[i],i);    // 目的関数
    sum(a[i]*x[i],i) <= b;     // 制約条件
```

と定義されたモデルから生成された knap について，VC++の GUI で　“.” を打った後に以下のように，参照可能なメンバーが現れますが，それぞれはこのモデル中のオブジェクトに対応しています.



例えば

```
knap.x
```

はモデル中の変数（Variable x）に対応し，このオブジェクトに対する表示手続き：

```
print(),cout,simple_printf()
```

は通常の SIMPLE オブジェクトに対するのと全く同様に可能です．すなわち

```
knap.x.print();
simple_printf(“objective = %d¥n”,knap.obj);
```

とすると，それぞれ knapsack.smp のモデル中の x,obj の値が表示されます.

```
simple_printf(“x[%s] = %d¥n”,knap.i,knap.x[knap.i]);
```

は少々複雑ですが，”knap.i” はモデル中の Element i の意味となりますので，モデル中で

```
simple_printf(“x[%s] = %d¥n”,i,x[i]);
```

としたのと同じ，変数と添字の書式付表示となります.

5.2 C/C++の配列の内容の設定

モデルを操作するルーチンが行う最初の操作は C/C++の配列として渡されたデータを SIMPLE のオブジェクトに設定するということです．スカラ値の場合には，double または int の値をそのまま

```
Parameter b;
 b = barg; // barg は double の入力引数
```

と代入できますが，添字付けられた大量のオブジェクトを配列から一気に読み込むには readD という手続きを使います．readD は一般に C の配列から SIMPLE のオブジェクトを設定するためのツールで，本章のようなアプリケーション接続の際によく利用されます．呼び出しの一般形は次の通りです．

```
void readD(int len1,int len2,..,int lenM,double* data) const
```

引数の型：

len1,len2,…,lenM : 次元のサイズ

data : 実際のデータ

与えられる配列データ（data）を一般の多次元配列（添字は１から始まる整数値）に読みかえて解釈するので，行列など多次元の配列を定義することができます．次は実際の利用例です．

```
// 配列内容
      double cont[27]={1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7,
                       2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7,
                       3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7,
                       4.1, 4.2, 4.3, 4.4, 4.5, 4.6};
      Set a1(name="a1");  Element i1(set=a1);
      Parameter p1(index=i1);   // １次元配列データ
      p1.readD(5, cont);
      // 結果: p1: [1]=1.1,[2]=1.2,[3]=1.3,[4]=1.4,[5]=1.5

      Set a2(name="a2",dim=2); Element i2(set=a2);
      Parameter p2(index=i2);  // ２次元配列データ
      p2.readD(2, 3, cont);
      // 結果: p2: [1,1]=1.1,[1,2]=1.2,[1,3]=1.3
      //           [2,1]=1.4,[2,2]=1.5,[2,3]=1.6

      Set a3(name="a3",dim=3); Element i3(set=a3);
      Parameter p3(index=i3);  // ３次元配列データ
```

```
p3.readD(2, 3, 2, cont);
// 結果: p3: [1,1,1]=1.1,[1,1,2]=1.2,[1,2,1]=1.3,[1,2,2]=1.4
//           [1,3,1]=1.5,[1,3,2]=1.6,[2,1,1]=1.7,[2,1,2]=2.1
//          [2,2,1]=2.2,[2,2,2]=2.3,[2,3,1]=2.4,[2,3,2]=2.5
```

5.3 求解

データの設定が済んだら数理計画問題を解きますが，loadmodule プロジェクトの driver.cpp (BC++/UNIX 版では useClass.cc)のように，genClass で生成したクラス定義から作ったモデルを解いている場合には宣言を

```
System_knapsack knap(c,a,b); // knapsack 問題の求解
```

と宣言を行えば自動的に求解が行われます. 変数の初期値の設定を行ってから求解したい場合などは

```
options.noDefaultSolve = 1;
```

と設定してからオブジェクトの宣言を行い,

```
knap.x[i] = 1; // 初期値の設定
knap.solve(); // ここで求解
```

と solve() という手続きを呼びます.

noClass プロジェクトの noClass.cpp (BC++/UNIX 版では MIP.cc)の場合のようにモデル定義と操作を同時に行っている場合には

```
solve(); // 最適化の実行
```

とすれば，それ以前に定義したモデルに対する求解が行われます.

5.4 C の配列への書き出し

操作する手続きが次に行うのは，求解した結果を C++ の配列の形で取得することです．システムオブジェクトに対して dump() という手続きを起動します．loadmodule プロジェクトの driver.cpp(BC++/UNIX 版では useClass.cc)では

```
int len;
char** ind;
double* knapx;
knap.x.val.dump(len,ind,knapx);
```

と行うことによって，C++の配列 knapx に解を書き出しています．一般に dump の引数は

dump(int len,char*& ind,double* data);

で，

len: ind,data の総長さ
ind: インデックス文字列の配列（長さ len）
data:データ本体 （長さ len）

です．data[0],data[1],…には添字 ind[0],ind[1],…がそれぞれ対応しています．データの並び順はインデックスをソートした際の自然な順番（数字の場合には昇順，文字列の場合には辞書順）となり，この場合，入力データは readD によって設定されているので，データ並びに対応する knapx には番号順に値が設定されることが保証されます．そのため，driver.cpp (BC++/UNIX 版では useClass.cc)ではインデックスを見ないで戻りの配列に値を設定しています．なお，ind,data に対応する領域は dump 内部で確保されるためコールした側で解放する必要があります．スカラー値（添字のないオブジェクト）は長さ 1 のオブジェクト（添字の値としてはヌル文字列が設定されます）として同様に dump で取得することができますが，便利な手続きとして double を返す asDouble()という手続きがあり，

```
double objval = obj.asDouble();
```

のように値を取ることができます．

5.5 計算結果に関する情報の取得

大域変数である result に直前の最適化の結果が設定されます．よく利用されるのは

```
int result.status  … 最適化の終了時の状態
```

で，最適化のステータスコード（正常終了時には0）を返します．

```
double result.optValue;
```

は最適化後の目的関数の値で，dump() による取得よりも簡便なので，ここでは目的関数値を戻すのに利用しています．

5.6 求解操作と代入

VariableParameter の値を変化させながら複数回最適化を行う場合，モデルに対応するシステムオブジェクト model について行う場合には

```
for ( int p = 1; p <= 5 ; ++p ) {
        model.p = p; // VariableParameter の設定
        model.solve();// 求解
}
```

のように記述を行います．(p がモデル中の VariableParameter の名前とします)．システムオブジェクトのメンバ (上記では p) に対しての代入は通常のモデル内での代入と同様に機能します．

noClass プロジェクトの noClass.cpp の場合のようにモデル定義と操作を同時に行っている場合には

```
for ( int pi = 1; pi <= 5 ; ++pi ) {
        p = pi; // VariableParameter の設定
        solve();// 求解
}
```

とします(VariableParameter p と同じ名前の int 変数 p を使うことはできませんので，pi としています)．

5.7 NUOPT オプション

プログラム内で NUOPT の動作を制御するパラメータを設定することができます. solveLP,solveQP を用いて問題を解く際に NUOPT のパラメータを設定するには

```
nuoptParam param;
```

のように NUOPT のパラメータ群を示す nuoptParam なるクラスのオブジェクトを定義して（名前は任意),

```
param.パラメータ名 = 値
```

として, パラメータを設定, その nuoptParam のオブジェクトを solveLP,solveQP の最初の引数として与えます. こうすると，その問題を解く際（solveLP,solveQP のコール）に対してここで設定したパラメータが与えられます.

SIMPLE モデルに対してパラメータを設定するには，モデルやドライバ中に

```
options.パラメータ名 = 値;
```

と書きます. ”options”は大域的に有効な nuoptParam のクラスオブジェクトです. 設定した内容は次の NUOPT の起動時に反映されます. ”options”は大域的に有効なので, 以前の指定が残ることにご注意ください.

nuoptParam クラスを使って定義できるクラスオブジェクトには以下があります.

名称	選択	Default	意味
outputMode	"silent", "normal",	"normal"	標準出力モード [output:mode = normal]
method	"auto", "line", "higher", "trust", "bfgs", "simplex", "asqp", "lsqp", "tsqp"	"auto"	求解アルゴリズム [method:auto]
scaling	"off" "on"	"on"	スケーリングを行うか否か [scaling:on]

maxitn	int	150	内点法の反復回数の最大 [crit:maxitn = 150]
eps	double	自動設定	内点法の停止条件(内点法専用) [crit:eps = 1.0e-8]
addToCutoff	double	0	足切り点設定用パラメータ (分枝限定法専用) [branch:addtocutoff = 0]
cutoff	double	未定義	足切り点 (分枝限定法専用) [branch:cutoff = 1.8]
p	int	10	探索深さ (分枝限定法専用) [branch:p = 10]
maxnod	int	-1（無制限）	探索問題数上限 (分枝限定法専用) [branch:maxnod=100000]
maxtim	int	-1（無制限）	計算時間上限(秒) (分枝限定法と内点法全般[8]) [branch:maxtim=3600]
maxmem	int	-10（UNIX：無制限 Windows:10Mb）	分枝限定法のメモリ利用量上限（Mb），残り利用可能メモリによる制限[9]（Windows版のみ，負値，Mb） [branch:maxmem=500]
gaptol	double	-1（指定なし）	上下界ギャップの下限（この値を下回ったら停止）
tolx	double	1.0e-8	主問題の実行不可能性判定値（単体法のみ） [param:tolx=1.0e-8]
told	Double	1.0e-6	双対問題の実行不可能性判定値（単体法のみ） [param:told=1.0e-6]

[8] Ver5以降より，内点法全般（higher/linear/bfgs/trust）に対してもこのパラメータが有効になりました.

[9] maxmemに負の値を設定すると，システムの利用可能なメモリが-maxmem[Mb]を切ったら実行を停止します（Windows版のみの機能）．デフォルト値-10はUNIX版では無制限という意味ですが，Windows版では１０Mbを切ったら実行を停止するという意味です.

maxintsol	int	-1（無制限)	整数解取得個数上限（秒） [branch:maxintsol=3]
mipfeasout	"off" "on"	“on”	分枝限定法の整数解発見についてのレポートを行うかどうか[10] [branch:mipfeasout=on]
outfilename	char*	0(未定義)	NUOPT の解ファイル名 [output:name=myout]
noDefaultSolve	int	0	solve()を陽に呼ばないと求解を行わない
noDefaultSolout	int	0	Solout()を陽に呼ばないと解の出力を行わない
outputParameter outputSet outputElement outputExpression	int	0 0 0 1	Parameter,Set,Element,Expression の CSV ファイル出力を行うかどうか
multDataPolicy	int	0（許さない)	同一のデータについてデータを重複して与えることを許すかどうか（１に設定すると警告扱いとなる）

[10] Ver6 以降より分枝限定法の標準出力が詳細になったのに伴って，整数解の情報は常に出力されるため，このパラメータ指定は意味を失っています．

6. VC++プロジェクトの設定

6.1 Microsoft Visual Studio 2008 プロジェクトの設定

ソリューション

nuopvcapp

に追加されているプロジェクトでは設定済みですが，一般に NUOPT のライブラリとの連結を行ってライブラリ（.LIB），あるいは DLL を作成する場合には，プロジェクトの設定を行う必要があります．

それでは，Visual Studio 2008 から NUOPT を利用する際に行う必要のある設定を紹介します．

- まず，プロジェクトの構成を「 Release」にする必要があります．下図のように，メニューバー下部にあるコンボボックスに「 Release 」と設定します．

図 1「プロジェクトの構成」の設定

これより以下の設定は，プロジェクト→プロパティ→構成プロパティで行います．

- 「構成プロパティ」→「C/C++」→「コマンドライン」と選択し，ウインドウの下部ある「追加のオプション」に

  @"C:¥Program Files¥nuopt¥bin¥cflags.cfg"

  を追加します．下線部は NUOPT のインストール場所に対応します．

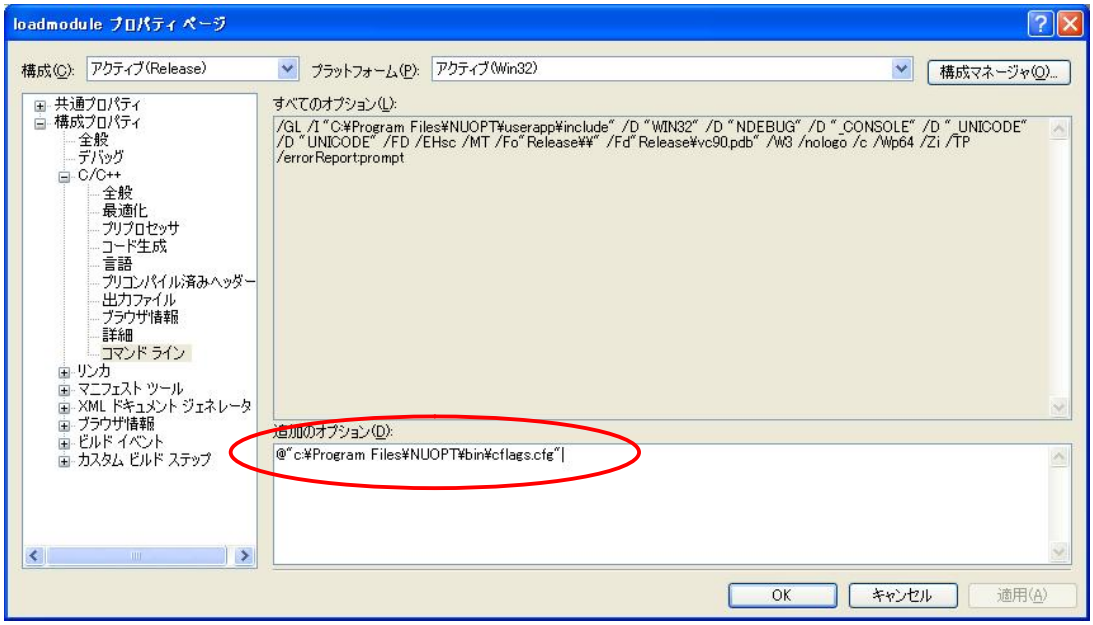


図 2：「追加のオプション」の設定

◆ 「構成プロパティ」→「C/C++」→「全般」と選択し，ウインドウ内にある「追加のインクルードディレクトリ」に

C:¥Program Files¥nuopt¥userapp¥include

を追加します．なお，下線部は NUOPT のインストール場所に対応します．

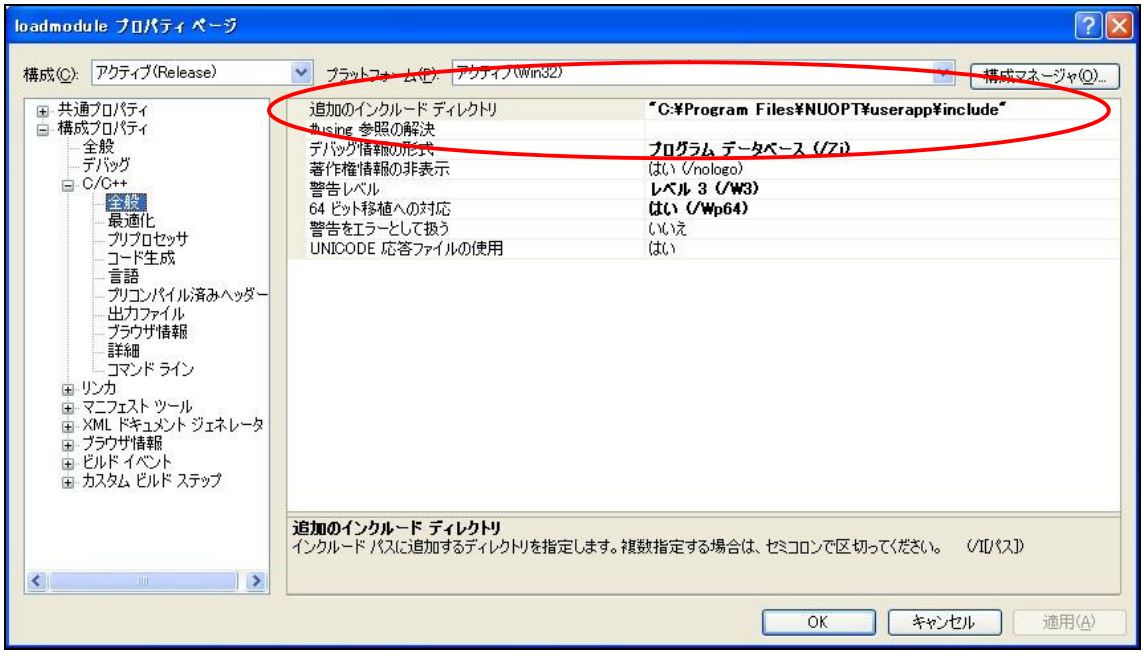


図 3：「追加のインクルードディレクトリ」の設定

- 「構成プロパティ」→「リンカ」→「入力」と選択し，ウインドウ内にある「追加の依存ファイル」に

```
libnuopt_MT_m.lib, psapi.lib
```

を追加します[11].

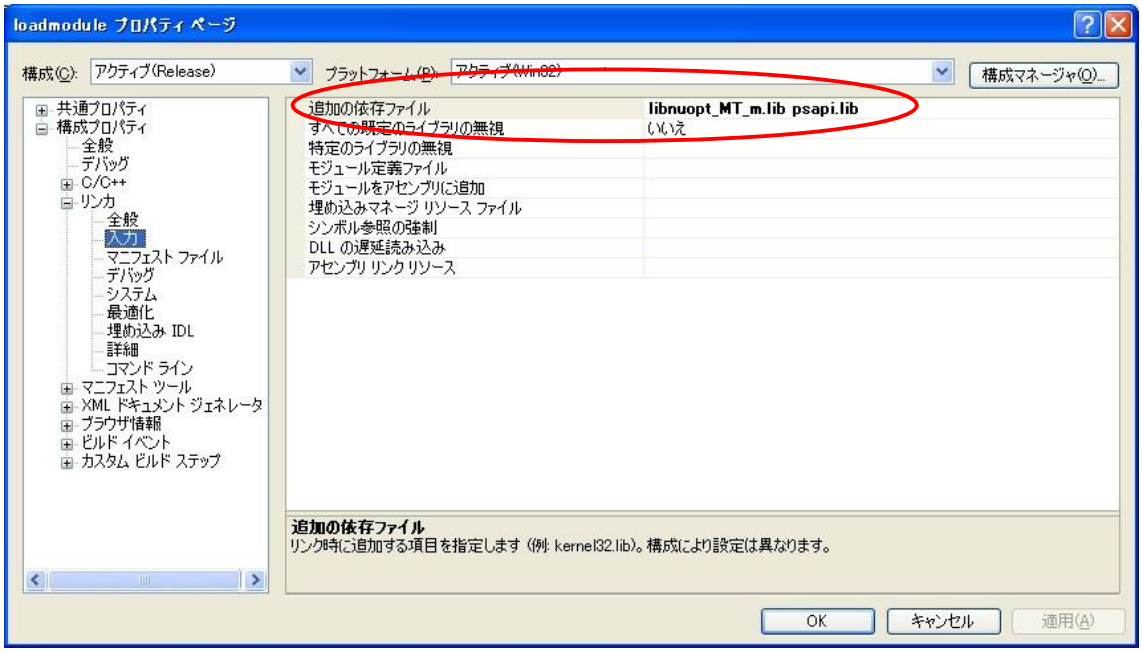


図 4：「追加の依存ファイル」の設定

- 「構成プロパティ」→「リンカ」→「全般」と選択し，ウインドウ内にある「追加のライブラリディレクトリ」に

```
C:¥Program Files¥nuopt¥userapp¥lib
```

を追加します．下線部は NUOPT のインストール場所に対応します．

[11]本設定は NUOPT のライブラリに接続しているアプリケーションで使用するライブラリがデフォルトの「マルチスレッド」でコンパイルする場合です．詳細は（注意１）をご覧下さい．

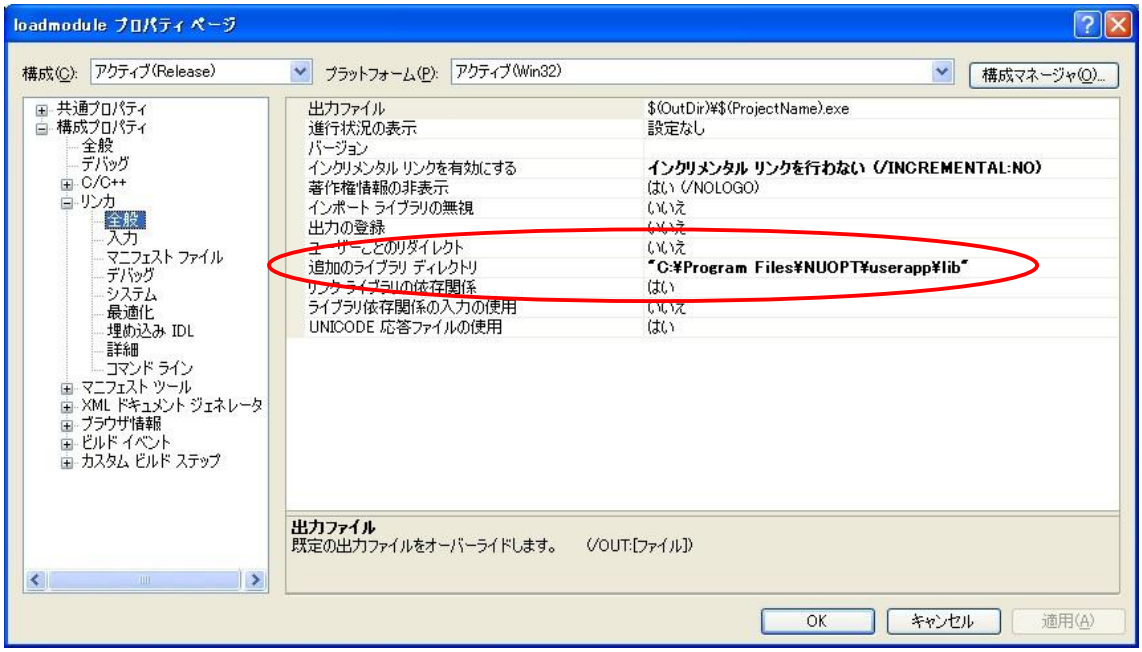


図 5：「追加のライブラリディレクトリ」の設定

◆ 注意1(アプリケーションで利用しているランタイムライブラリについて)：

上記は NUOPT のライブラリに接続しているアプリケーションで使用するランタイムライブラリが「マルチスレッドスレッド」でコンパイルされている場合です（この設定はプロジェクト→プロパティ→C/C++→コード生成で現れる「ランタイムライブラリ」の選択に対応します）．もし，NUOPT のライブラリと接続するコードがデフォルトと違って以下のようになっている場合には，３．で指定するライブラリの名前を「マルチスレッド DLL」なら

```
libnuopt_MD_m.lib
```

としてください．

◆ 注意2(マシン種別に応じた高速化について)：

また，大規模問題を解く場合などには，マシン種別に依存したライブラリを利用することによって速度の向上が図れる場合があります．その場合には別の設定方法を行う必要がありますので，

nuopt-support@msi.co.jp

までご連絡ください．

6.2 Microsoft Visual C++ 6 プロジェクトの設定

まず, NUOPT の添付例には, nuoptapp.sln が付属していますが, これは Visual Studio 2008 用のソリューションです. VC.NET では nuoptvcapp.sdw がこれに対応しますので, こちらをご利用下さい. ワークスペース

nuopvcapp

に追加されているプロジェクトでは設定済みですが，一般に NUOPT のライブラリとの連結を行ってライブラリ（.LIB)，あるいは DLL を作成する場合には，プロジェクト→設定で次の設定を行う必要があります.

1. 「プロジェクトオプション」のウインドウの先頭に

@"c:¥Program Files¥NUOPT¥bin¥cflags.cfg"

を追加します．下線部は NUOPT のインストール場所に対応します．この設定がないと

ｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙがｵｰﾌﾟﾝできません．'strstream.h':

というエラーが現れます.

2. C/C++タブ，カテゴリを「プリプロセッサ」にして「インクルードファイルのパス」を

c:¥Program Files¥NUOPT¥userapp¥include

下線部は NUOPT のインストール場所に対応します．この設定が誤っていると

ｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙがｵｰﾌﾟﾝできません．'simple.h'

というエラーが現れます.

3. リンクタブ，カテゴリを「インプット」にして

「オブジェクトライブラリモジュール」の先頭に

libnuopt_ML_m.lib

を追加します.

この設定が存在しないとリンクの際に

SystemInterface::SystemInterface(char const *)

などの未解決エラーになります．また,

_memmove はすでに LIBCD.lib(memmove.obj) で定義されています

などのエラーが多数出力される場合には, 以下の注意の項をご覧ください. アプリケーションで利用しているランタイムライブラリの設定の問題である可能性があります.

4. さらに「追加ライブラリのパス」を

c:¥Program Files¥NUOPT¥userapp¥lib

にします.

下線部は NUOPT のインストール場所に対応します．この設定が誤っていると

ﾌｧｲﾙ "libnuopt_ML_m.lib" を開けません.

というエラーになります.

以上の設定を終えたら，ビルドが可能です.

◆ 注意1(アプリケーションで利用しているランタイムライブラリについて):

ただし上記は NUOPT のライブラリに接続しているアプリケーションで使用するランタイムライブラリがデフォルトの「シングルスレッド*」あるいは「シングルスレッド（デバッグ)」でコンパイルされている場合です（この設定はプロジェクト→設定の「C/C++タブ」で「コード生成」で現れる「使用するランタイムライブラリ」の選択に対応します)．もし，NUOPT のライブラリと接続するコードがデフォルトと違って以下のようになっている場合には，３．で指定するライブラリの名前を

「マルチスレッド」あるいは「マルチスレッド（デバッグ)」なら

libnuopt_MT_m.lib

「マルチスレッド DLL」あるいは「マルチスレッド DLL（デバッグ)」なら

libnuopt_MD_m.lib

としてください.

◆ 注意2(マシン種別に応じた高速化について):

また，大規模問題を解く場合などには，マシン種別に依存したライブラリを利用することによって速度の向上が図れる場合があります. その場合には別の設定方法を行う必要がありますので,

nuopt-support@msi.co.jp

までご連絡ください.

◆ 注意3(無害な警告について):

Debug / Release の Debug を選ぶと以下の警告が出ますが，無害です.

LINK : warning LNK4098: defaultlib "LIBC" は他のﾗｲﾌﾞﾗﾘの使用と競合しています; /NODEFAULTLIB:library を使用してください

6.3 Microsoft Visual C++.net プロジェクトの設定

6.3.1 はじめに

「NUOPT 外部接続マニュアル」では，Visual Studio 2008 の GUI を元にした記述を行っている箇所があります．本節では，「NUOPT 外部接続マニュアル」にて問題になり得るであろう Visual Studio 2008 と VC.NET の GUI の違いを説明します．

6.3.2 VC++ プロジェクトの設定と実行

nuoptvcapp.sln をダブルクリックなどすると，VC.NET の GUI が起動します．

VC.NET の GUI から NUOPT を VC++ プロジェクトとして利用する際の設定内容は，以下のいずれかの操作によって表示されるウインドウから設定できます．

(a) GUI の右側にある「クラスビュー」ウインドウ内に各プロジェクトが表示されています．そのプロジェクト上で右クリックすると，**図 6** のようなメニューが表示されます．そのメニューの「プロパティ」を選択すると，設定ウインドウが表示されます．

(b) **図 7** のようにメニューバーから「プロジェクト」→「プロパティ」とすると，設定ウインドウが表示されます．

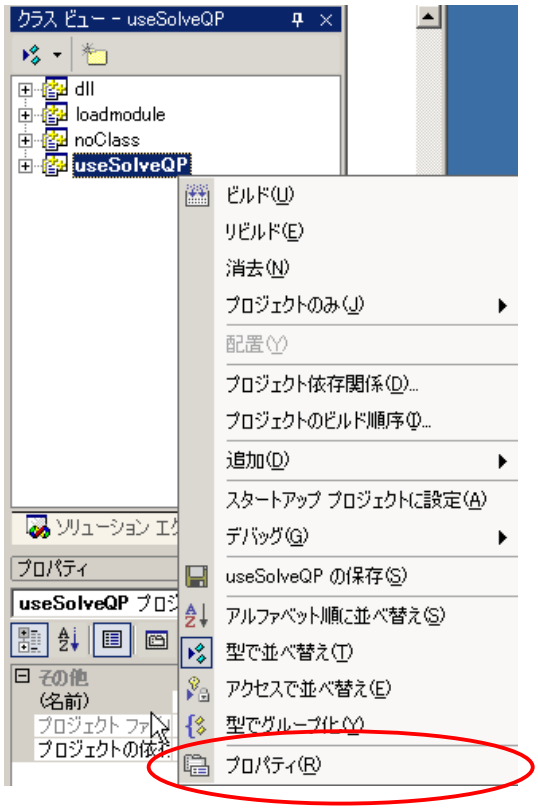


図 6：設定ウインドウの表示方法（1）

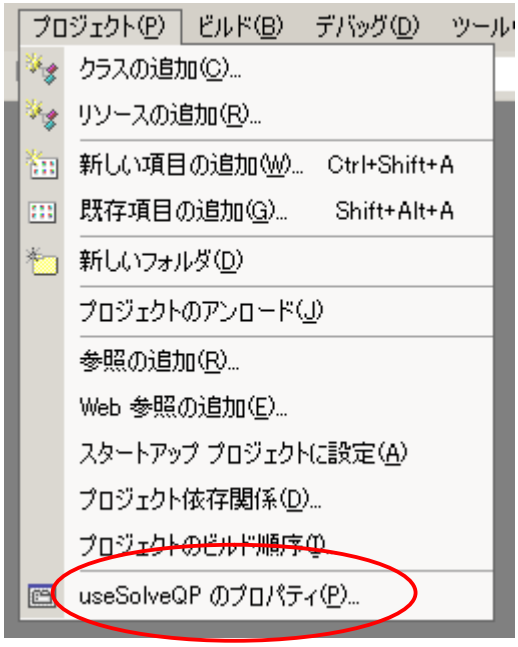


図 7：設定ウインドウの表示方法（2）

そのときに表示されるウインドウは**図 8** のようなものです.

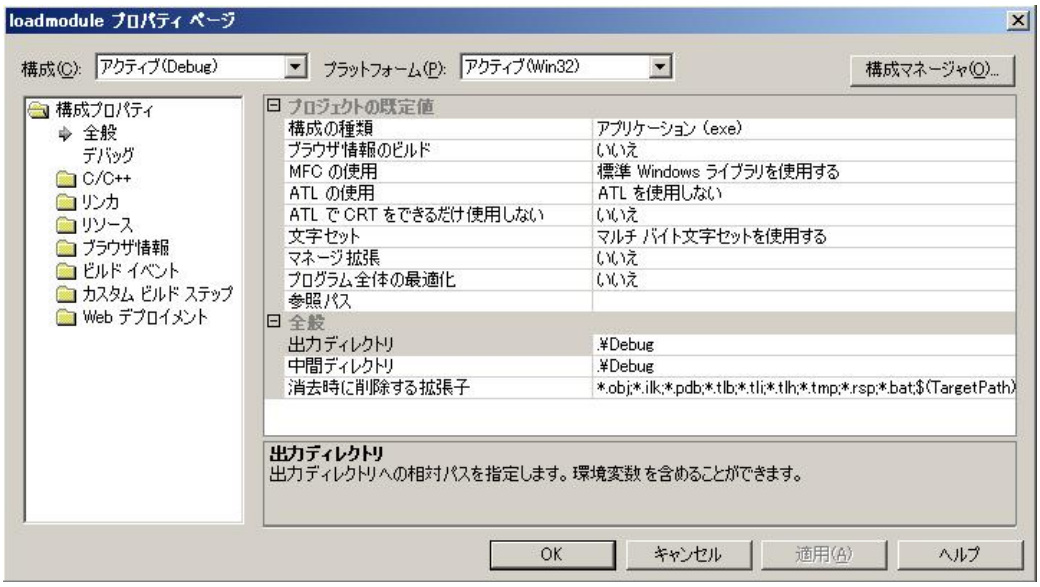


図 8：プロパティ設定ウインドウ

なお，(b) の場合に表示されるウインドウは，「スタートアッププロジェクト」に設定されているプロジェクトに対応する設定ウインドウです．「スタートアッププロジェクト」を設定するには，GUI 右側に表示されている「クラスビュー」内に表示されているプロジェクトのうち，「スタートアッププロジェクト」に設定したいプロジェクトの上で右クリックし，
図 9 のようなメニューを表示させます．

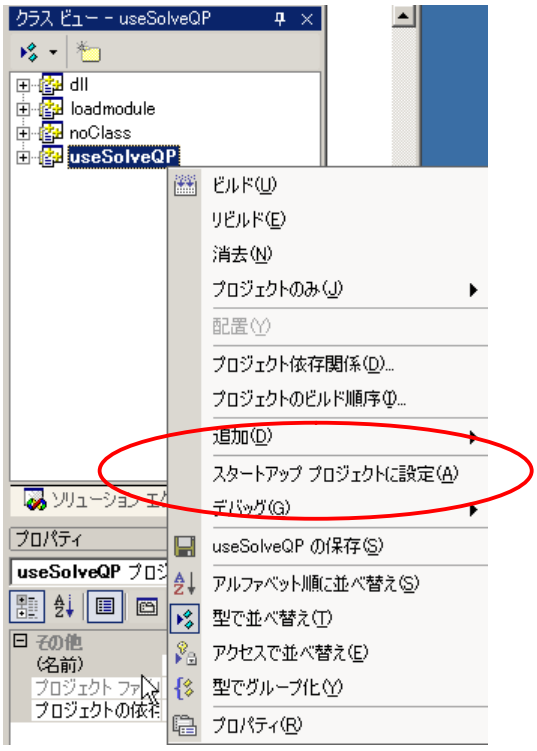


図 9：スタートアッププロジェクトの設定方法

図 9 のメニューから「スタートアッププロジェクトに設定」を選択すると，当該プロジェクトが「スタートアッププロジェクト」となります.

それでは，VC.NET から NUOPT を利用する際に行う必要のある設定を紹介します.

- 「構成プロパティ」→「C/C++」→「コマンドライン」と選択し，ウインドウの下部ある「追加のオプション」に

  @"C:¥Program Files¥nuopt¥bin¥cflags.cfg"

を追加します．下線部は NUOPT のインストール場所に対応します.

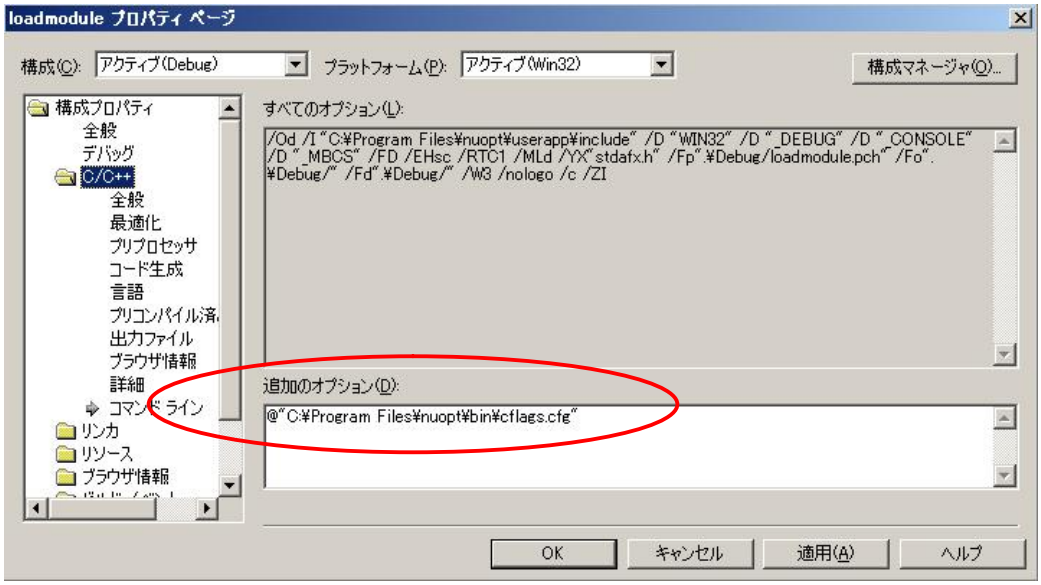


図 10 :「追加のオプション」の設定

- 「構成プロパティ」→「C/C++」→「全般」と選択し，ウインドウ内にある「追加のインクルードディレクトリ」に

  C:¥Program Files¥nuopt¥userapp¥include

を追加します. なお，下線部は NUOPT のインストール場所に対応します.

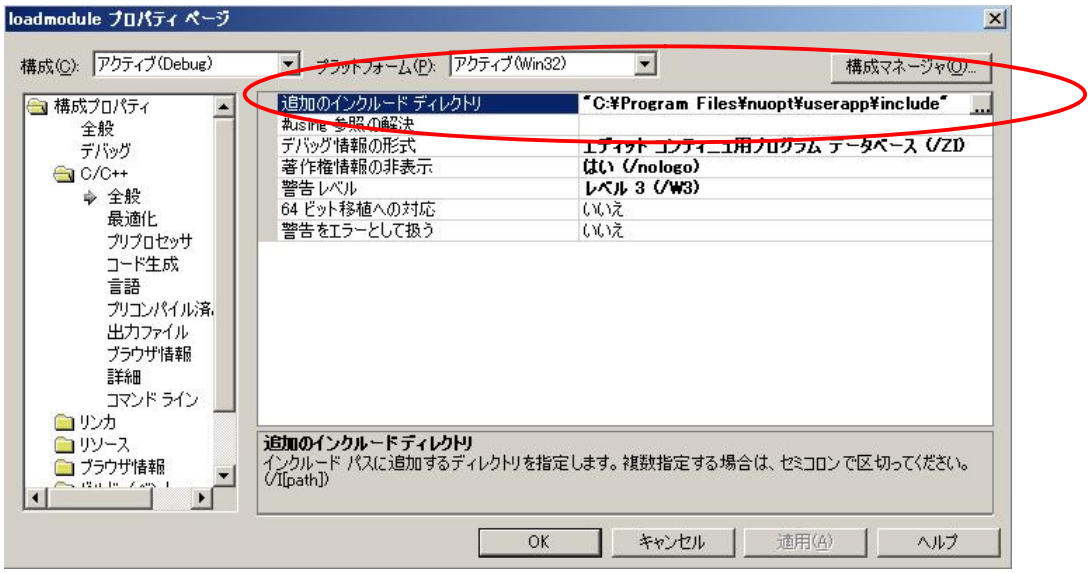


図 11：「追加のインクルードディレクトリ」の設定

- 「構成プロパティ」→「リンカ」→「入力」と選択し，ウインドウ内にある「追加の依存ファイル」に

```
libnuopt_ML_m.lib
```

を追加します[12].

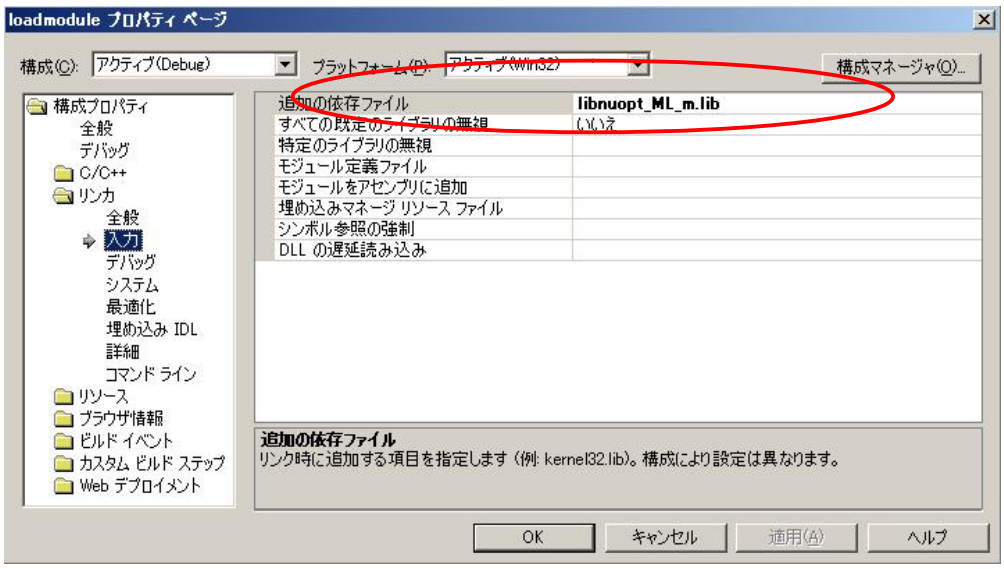


図 12：「追加の依存ファイル」の設定

- 「構成プロパティ」→「リンカ」→「全般」と選択し，ウインドウ内にある「追加のライ

[12]本設定は NUOPT のライブラリに接続しているアプリケーションで使用するライブラリがデフォルトの「シングルスレッド*」あるいは「シングルスレッド（デバッグ）」でコンパイルされている場合です．それ以外の場合には，本マニュアル 6.1 の（注意 1）をご覧下さい．

ブラリディレクトリ」に

C:¥Program Files¥nuopt¥userapp¥lib

を追加します．下線部は NUOPT のインストール場所に対応します．

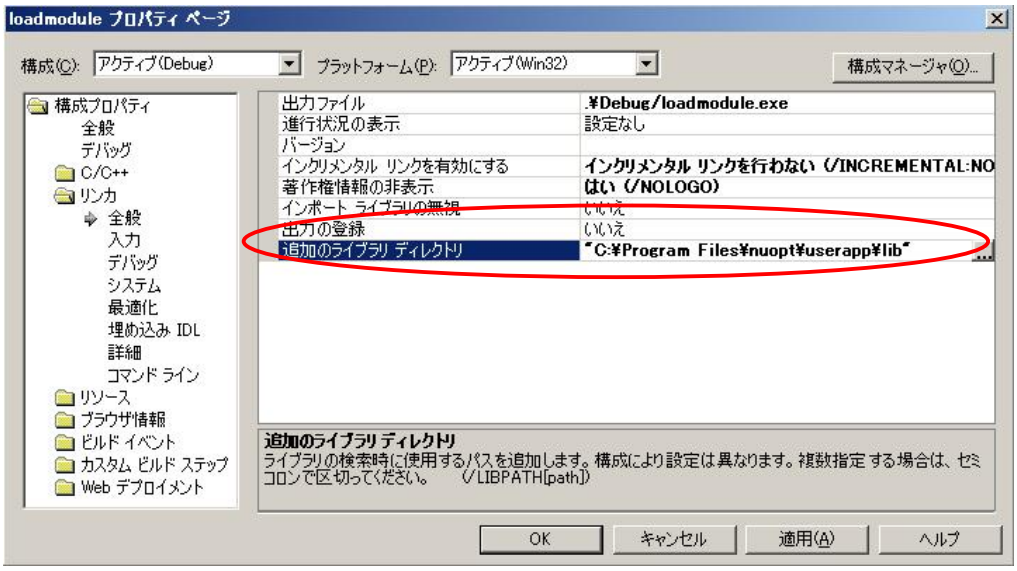


図 13：「追加のライブラリディレクトリ」の設定

これらの設定の後，ビルドが可能になります．ビルドを行う方法として，次の二つがあります．

- 「クラスビュー」にて当該プロジェクトを右クリックして表示されるメニューから「ビルド（リビルド)」を選択する．

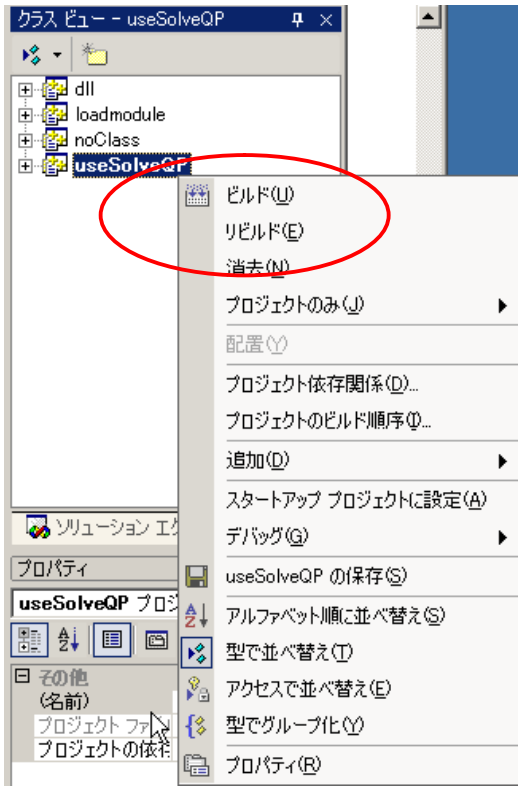


図 14：プロジェクトのビルド（1）

- 当該プロジェクトが「スタートアッププロジェクト」である場合には，メニューバーの「ビルド」から実行することもできます．

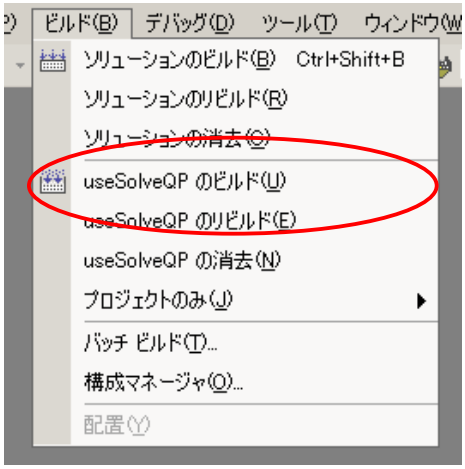


図 15：プロジェクトのビルド（2）

ビルドしたプロジェクトは，メニューバーの「デバッグ」から「開始（デバッグなしで開始）」を選択することで実行できます．

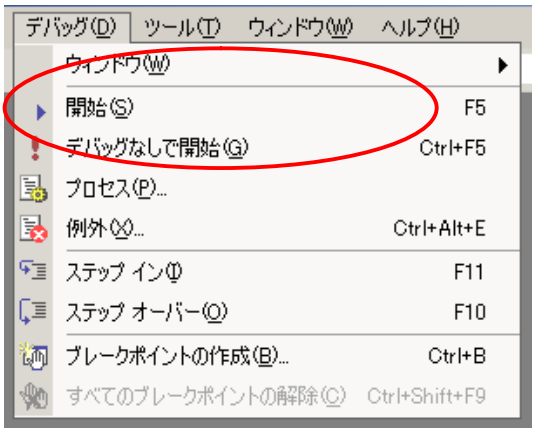


図 16：ビルドしたプロジェクトの実行

このとき，コマンドに引数を与える必要がある場合（NUOPT に付属の例ではプロジェクト 'useSolveQP' がこれにあたります）は，上記 (a) もしくは (b) にて表示されるウインドウにて「構成プロパティ」→「デバッグ」と選択し，そのウインドウに現れる「コマンド引数」にその内容を記述してください．

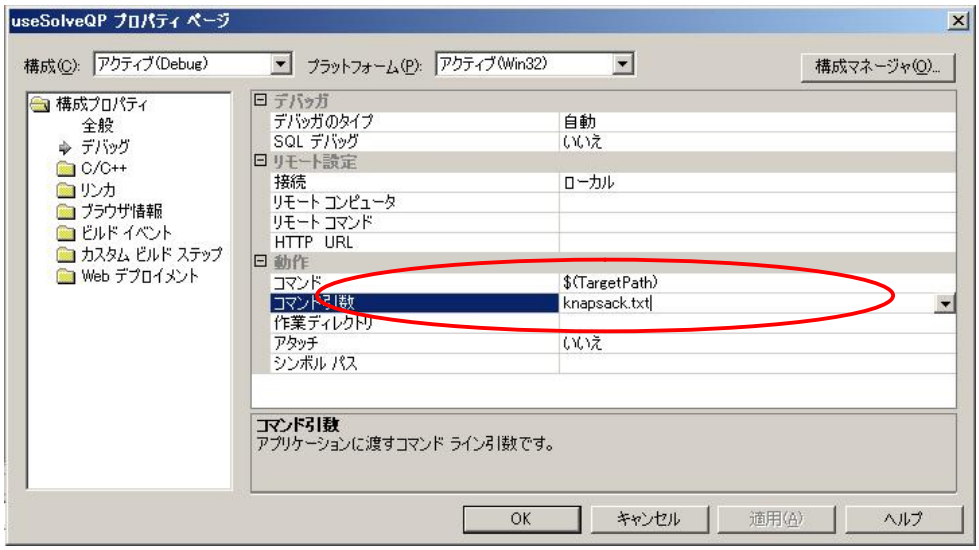


図 17：コマンドへ引数の設定

7．おわりに

本ドキュメントの内容についてのお問い合わせは

nuopt-info@msi.co.jp

までお寄せください．

索引

O

P

R

S

T

U

V

W

え

き

く

こ

し

せ

と

な

に

は

ふ

ま

め

よ

ら

る

れ