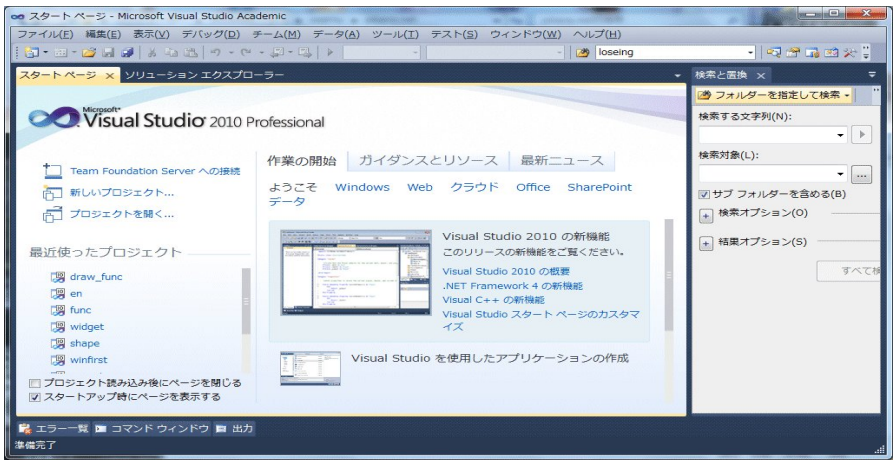
高知大学教育学部の情報数学のテキスト

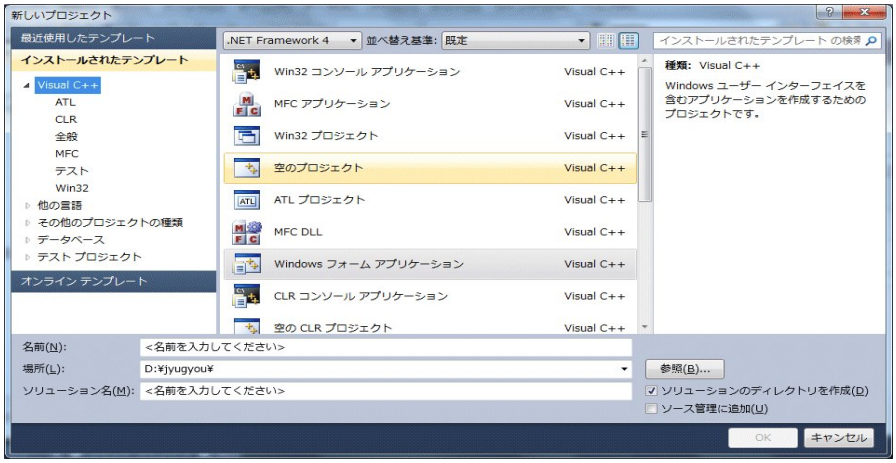
文責　：　高知大学名誉教授　中村　治

コンソール・プログラミング

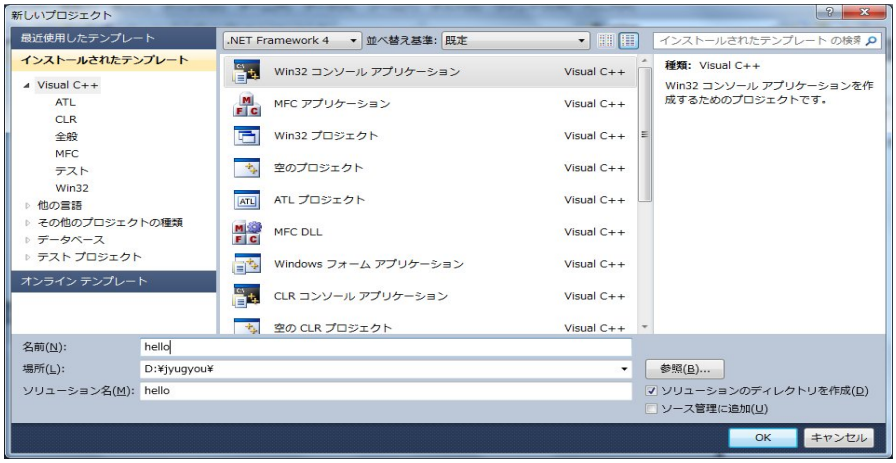
自分のためにちょこちょこと計算するには、コンソールアプリケーションを使います。
まず、Microsoft Visual Studio 2010 を立ち上げる。



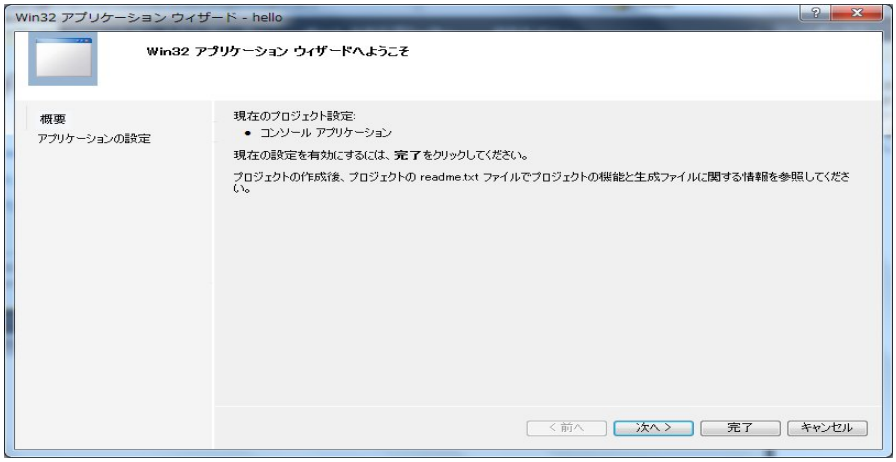
このようにならなければ、「表示」のメニューで「スタートページ」をクリックします。
新しいプロジェクト … をクリックする。



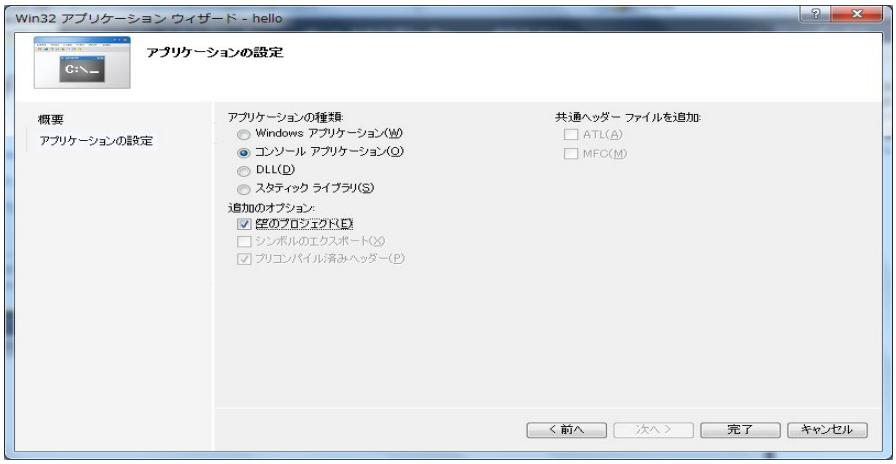
表示が違っていても気にしなくても良いです。Visual C++ になっていなければ、Visual C++ をクリック（選択）します。表示が違っていても気にしなくても良いです。「Win32 コンソール アプリケーション」が表示されていれば良いです。「Win32 コンソール　アプリケーション」をクリックし、図では場所 (L):が `D:\jyugyou\`となっています（自分で適当なディレクトリ、例えば `D:\jyugyou\`を作り、ここを指定すれば、プログラムが `D:\jyugyou\`に作られます）が、何処にプログラムが保存されるか気にしなければ、そのままで良いですので、名前 (N): のエディットボックスに例えば hello と入力し、



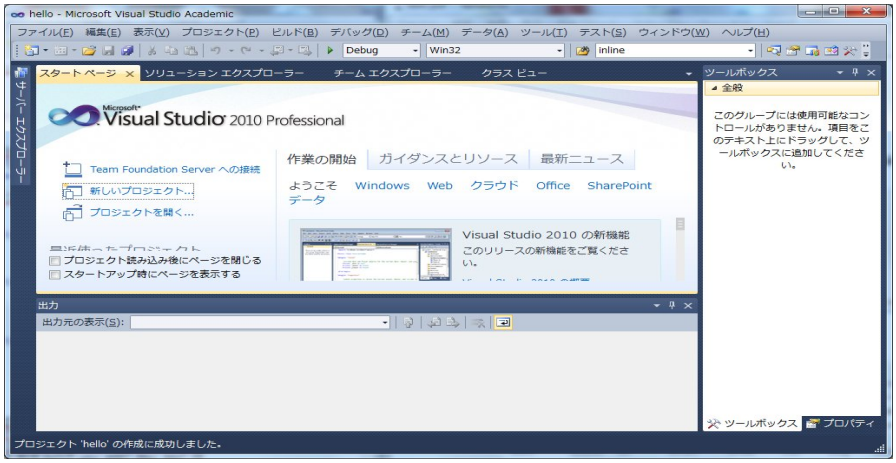
OK ボタンをクリックする。



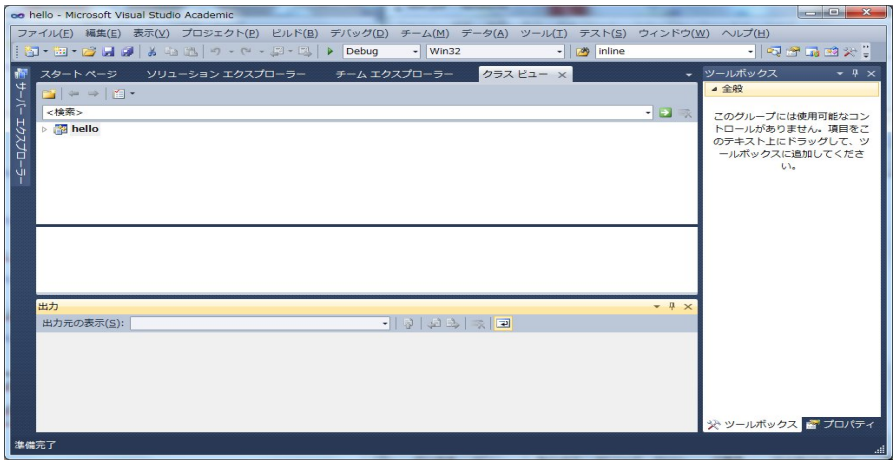
次へのボタンをクリックし、空のプロジェクトをチェックする。



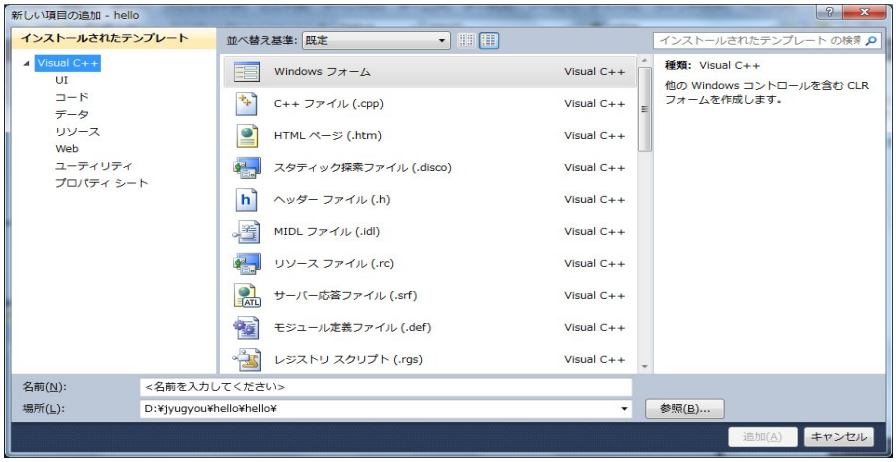
完了ボタンをクリックする。



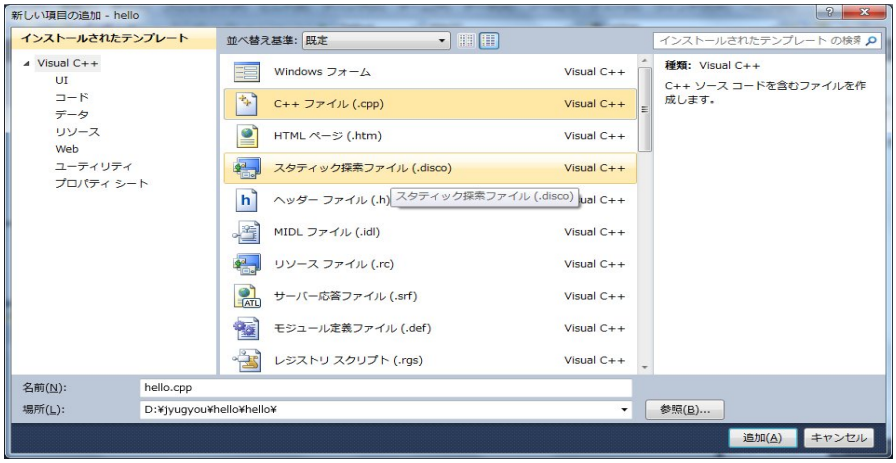
クラスビューになってなければ、クラスビューをクリックする。



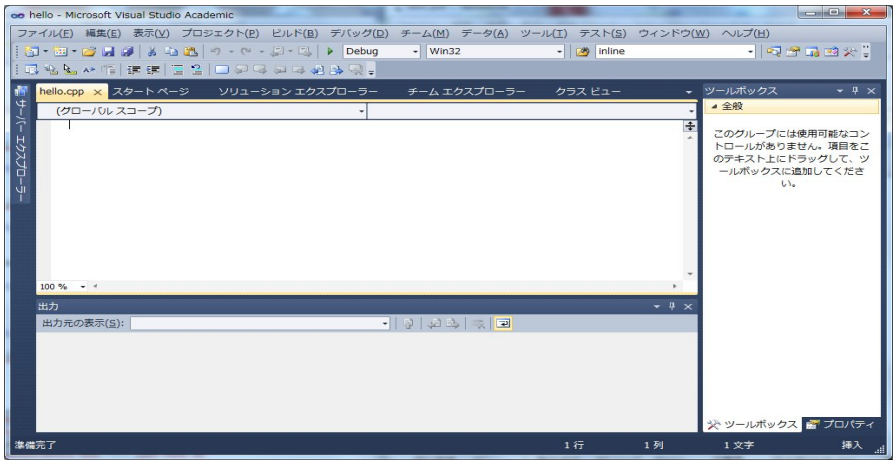
一段目の左から二つ目のアイコン（新しい項目の追加）をクリックする。



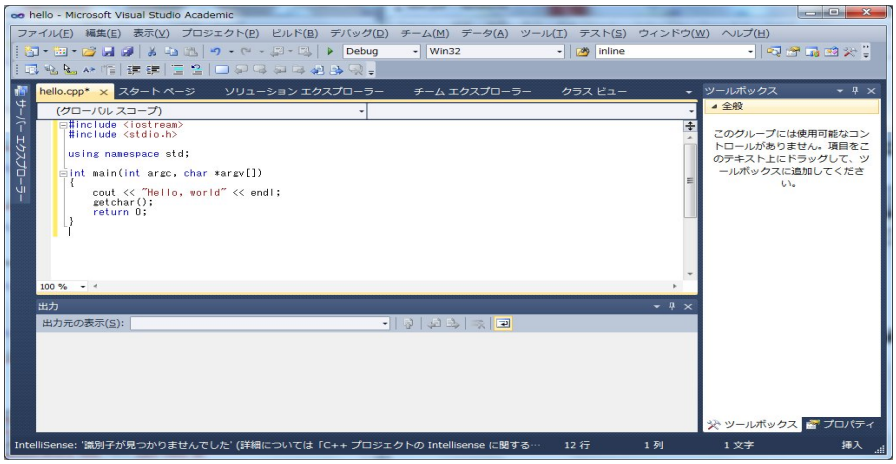
C++ ファイルをクリックし、名前の欄に hello と打ち込む。



追加のボタンをクリックする。



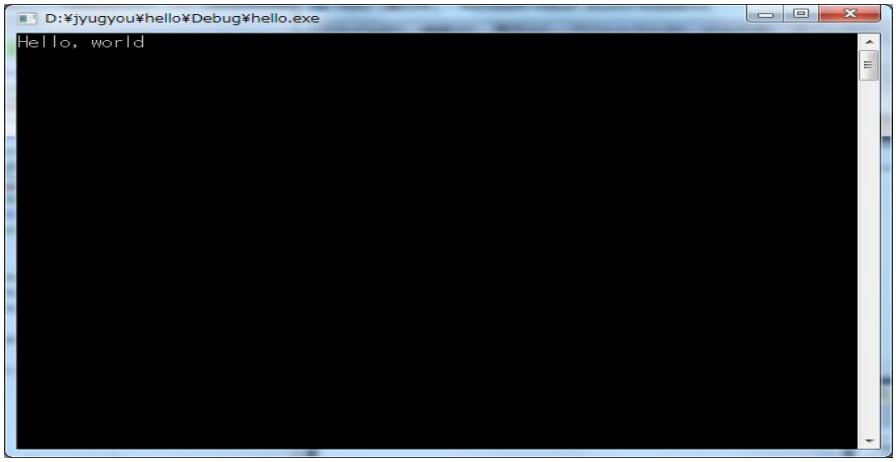
hello.cpp のファイルが作られ、開いている。ここにプログラムを打ち込む。




```
#include <iostream>

using namespace std;
```

```
int main(int argc, char *argv[])
{
        cout << "Hello, world" << endl;
        getchar();
        return 0;
}
```

コンパイルし、実行する（Debug の横の赤い三角をクリックする）と



と Hello,world と表示する。Enter キーを押すとウインドウが閉じます。

このプログラムを簡単に説明をします。

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        cout << "Hello, world" << endl;
        getchar();
        return 0;
}
```

の

```
int main(int argc, char *argv[])
{
        cout << "Hello, world" << endl;
        getchar();
        return 0;
}
```

が main() 関数の定義です。、コンピュタはこの main() 関数を一行目から順に実行します。ここで

```
int main(int argc, char *argv[])
{
}
```

が main() 関数の雛形で、一番最初の int で main() 関数が整数値を返す関数であることを宣言しています。次の main が関数の名前です。C++ のプログラムは必ずこの main() 関数が必要です。次 main() の (と) の間にある int argc, char *argv[] が main() 関数の引数で、二つの引数を取ります。引数はコンマ , で区切ります。最初の int argc は int でこの引数が整数値を取る引数であることを宣言し、次の argc がこの引数の名前です。次の char *argv[] は複雑な構造をしていて説明するのが厄介で、多くの初心者が挫折する char のポインタの配列を宣言していて、argv が引数の名前です。この講義ではこの引数を使ったプログラムは作らないので、これらの引数の意味は現時点では知らなくても良いです。C++ のプログラミングの勉強をしていくとそのうち理解できるようになります。無条件でこうするものだと初心者のうちは何も考えずにこのままプログラミングしてください。一度にすべてを理解することは出来ません。実際、コマンドラインの引数を使わないプログラムは次のようにこの引数を省略して

```
#include <iostream>

using namespace std;

int main()
{
        cout << "Hello, world" << endl;
        getchar();
        return 0;
}
```

というプログラムでも大丈夫です。

関数の本体は { と } で囲まれた部分に書きます。

一行目の

```
        cout << "Hello, world" << endl;
```

は文字列　Hello, world をディスプレイに表示し改行しなさいと言う命令です。 ”Hello, world” のように” と ” で文字列を囲むことで C++ にこれは文字列であることを知らせています。

```
        cout <<  ???
```

の形で ??? が評価され、ディスプレイに表示されます。この行は

```
        cout << "Hello, world";
        cout << endl;
```

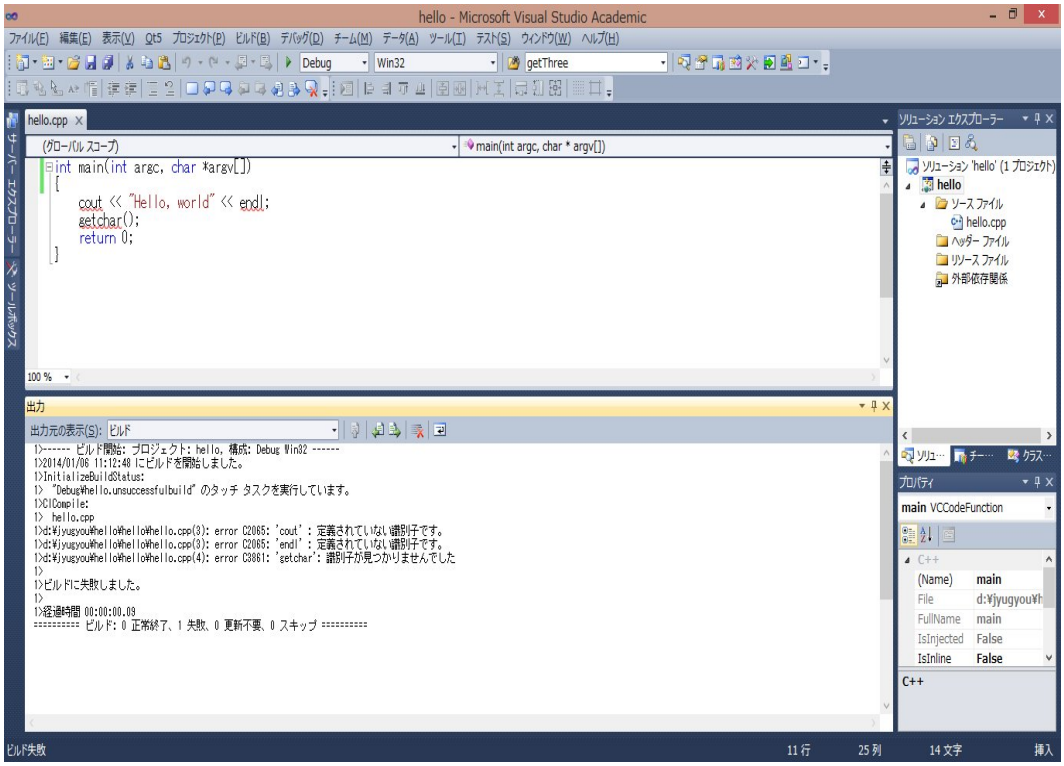
という表現を一行で表したものです。endl は C++ の約束でこの表現で改行します。C++ の文は必ず最後はセミコロン ; で終わります。ここで、プログラムを

```
int main(int argc, char *argv[])
{
        cout << "Hello, world" << endl;
        getchar();
        return 0;
}
```

として、コンパイル・実行すると

```
1>d:\jyugyou\hello\hello\hello.cpp(8): error C2065: 'cout' : 定義されていない識別子です。
1>d:\jyugyou\hello\hello\hello.cpp(8): error C2065: 'endl' : 定義されていない識別子です。
1>d:\jyugyou\hello\hello\hello.cpp(9): error C3861: 'getchar': 識別子が見つかりませんでした
```

というエラーメッセージが出ます。



cout と endl と getchar は知らないと言っています。これらがどう言うものかを書いているファイルを読み込みなさいという命令がプログラムの一行目の include 文です。

```
#include <iostream>
```

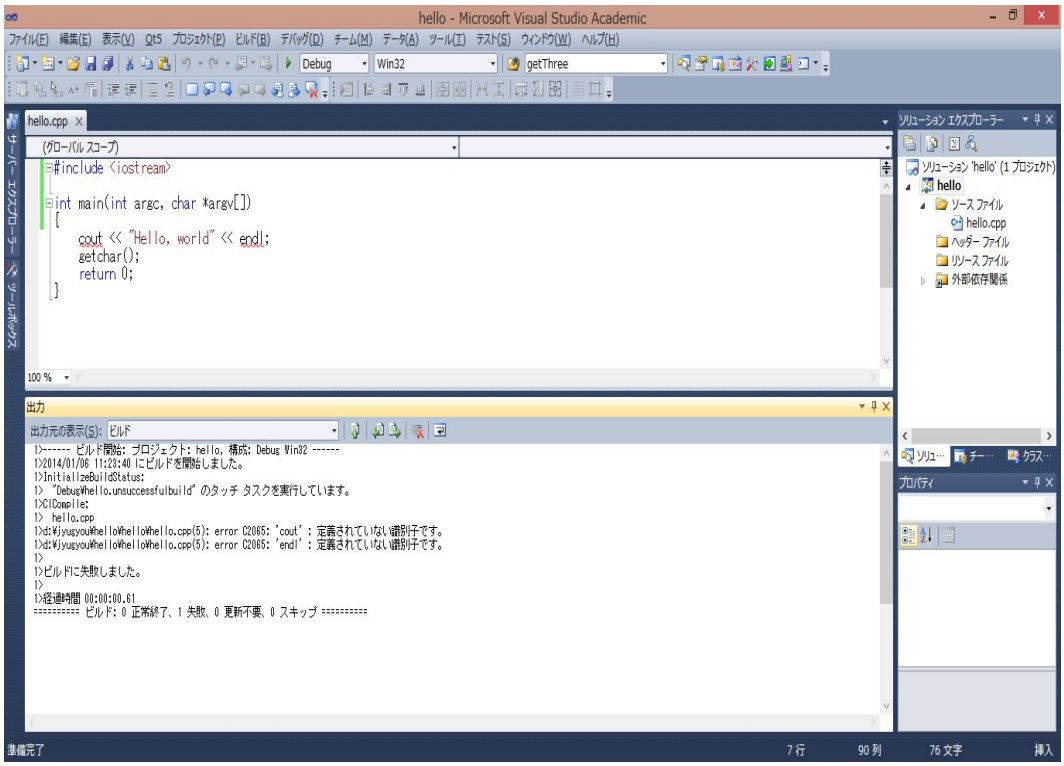
これが書式です。結果を表示しないプログラムはないので、C++ のプログラムはいつでもこの行を最初に書くものだと思って下さい。

```
#include <iostream>

int main(int argc, char *argv[])
{
        cout << "Hello, world" << endl;
        getchar();
        return 0;
}
```

として、コンパイル・実行すると

```
1>d:\jyugyou\hello\hello\hello.cpp(5): error C2065: 'cout' : 定義されていない識別子です。
1>d:\jyugyou\hello\hello\hello.cpp(5): error C2065: 'endl' : 定義されていない識別子です。
1>
1>ビルドに失敗しました。
```

というエラーメッセージが出ます。



getchar() は認識するようになりましたが、cout と endl とは相変わらず知らないと言っています。getchar() は C の関数なので認識してくれます。cout と endl は C にはなく、C++ で導入されたものです。これを認識させる方法は二通りあります。Python で

```
import math
```

とした時は、 math.sin(math.pi/6) としなければいけなかったのですが

```
from math import *
```

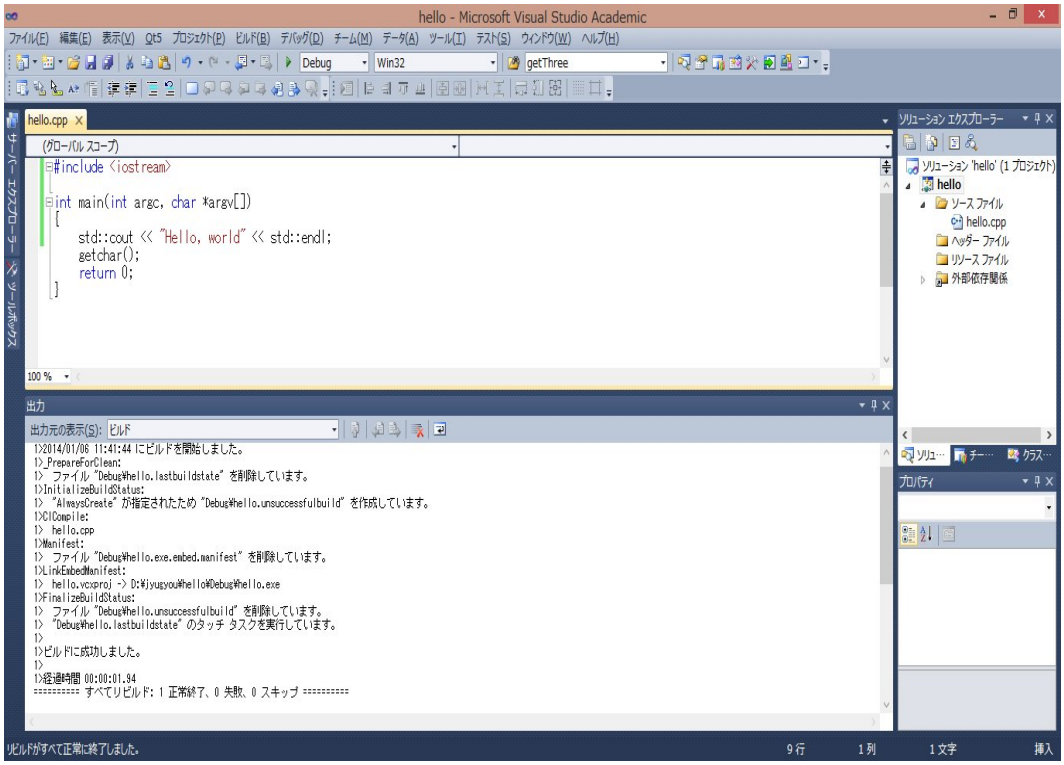
とした時は、 math. を取って sin(pi/6) で良かったです。今の状況は

```
import math
```

とした時と同じで、この場合は

```
#include <iostream>

int main(int argc, char *argv[])
{
	std::cout << "Hello, world" << std::endl;
	getchar();
	return 0;
}
```

と std:: という namespace の修飾語を付ければ認識するようになります。コンパイルします。



コンパイルできました。

いちいち std:: と書くのがめんどくさい時に

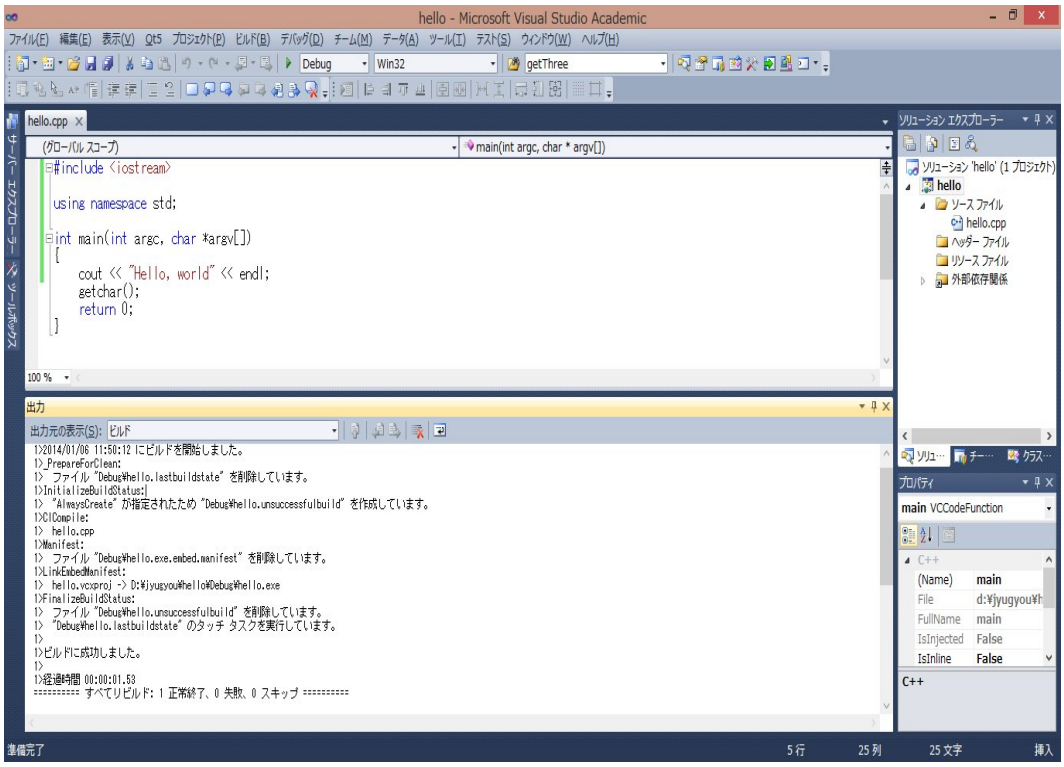
```
#include <iostream>

using namespace std;
```

```
int main(int argc, char *argv[])
{
        cout << "Hello, world" << endl;
        getchar();
        return 0;
}
```

のように二行目に

```
using namespace std;
```

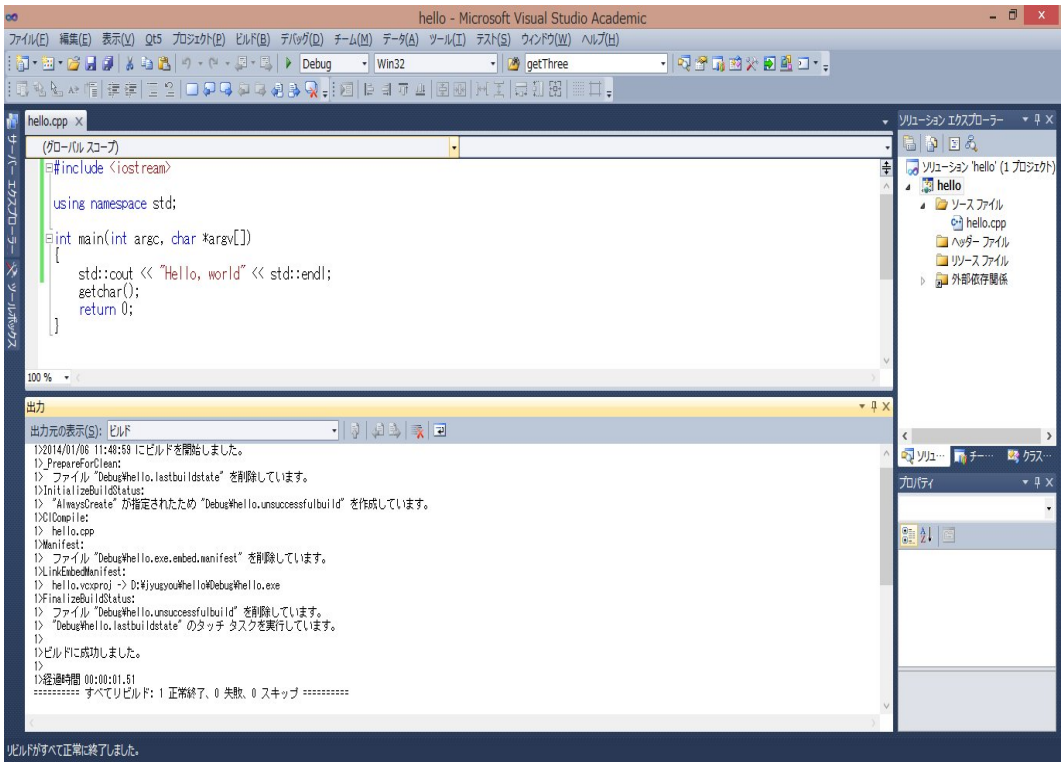
以下では、namespace std を使いますと宣言すれば、いちいち std:: と修飾語を付けなくても良くなります。コンパイルします。



コンパイルできました。

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        std::cout << "Hello, world" << std::endl;
        getchar();
        return 0;
}
```

と std:: の修飾語をつけていてもコンパイル・実行できます。コンパイルします。



コンパイルできました。
　コマンドプロンプトで実行するプログラムとしては

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        std::cout << "Hello, world" << std::endl;
        return 0;
}
```

で良いです。最後の行の

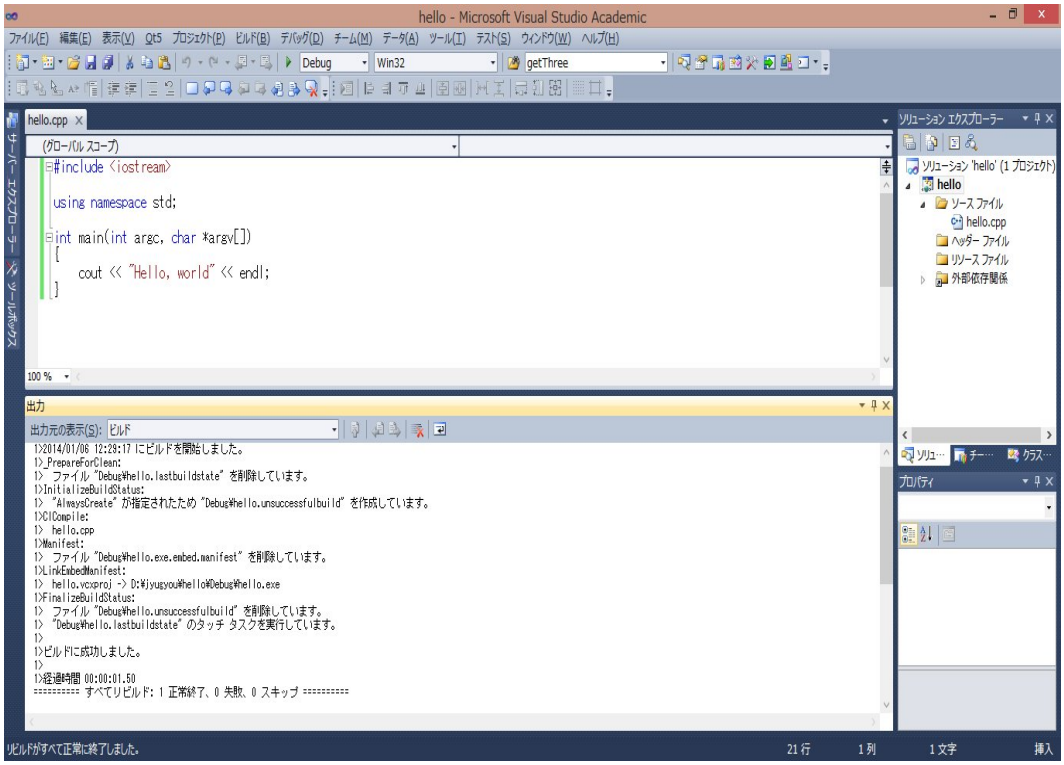
```
        return 0;
```

が return 文です。main() 関数が整数 int の値を返す関数であると宣言されているのでこの return 文を最後に書きます。実際は

```
#include <iostream>

using namespace std;
```

```
int main(int argc, char *argv[])
{
        cout << "Hello, world" << endl;
}
```

と main() 関数の return ぶんを省略しても VC++ はコンパイル・実行してくれます。



コンパイルできました。

main() 関数の場合は例外的にコンパイル・実行出来ますが、main() 関数が整数 int の値を返す関数であると宣言されているので、最後に return 文を書くのが普通です。

```
        return 0;
```

の 0 は 0 を main() 関数の値として返すと宣言しています。これは main() 関数が正常に実行出来たことを示しています。正常に実行できたという状態は一通りですが、エラーが起きたという場合は色々な場合があるので、正常に終了したときは 0 を返す約束になっています。この値は他のコンソールアプリケーションと連帯して使うときに必要になりますが、Windows のプログラミングではあまり使われませんし、この講義では単独で使うプログラムだけ作りますので、気にしなくて良いです。

最後に

```
        getchar();
```

ですが、これはキーボードから入力せれた 1 文字を受け取る関数です。これは VC++ の実行環境で実行したとき、プログラムの実行が終わると画面が消えてしまうのでそれを防ぐために、この命令をここに入れています。キーが押されたら、どのキーが押されたか、getchar() が捉えますが、その値は必要ないので、その値を受け取らずに関数だけ書いています。その値が何か知りたいときは

```
int c = getchar();
```

のようにします。この意味は今は気にしないでください。すぐ下で説明します。

以下では、このプログラムを雛形として

```
cout << "Hello, world" << endl;
```

の部分を作りかえて、プログラミングしていきます。

0.1 式と計算

いろいろな計算を行うには、変数と呼ばれるデータの記憶場所に数値を保存させる。変数は名前（変数名）を付けて区別する。変数名はアルファベット、数字、__（アンダーバー）からなる文字列で、先頭の文字には数字は使えない。大文字と小文字は区別するので abc ABC Abc AbC はすべて異なる変数を表す。次の識別子（名前）はキーワード（予約語）として使われるので、別の用途に使えない。予約語は文字がゴチックになるので、すぐわかります。

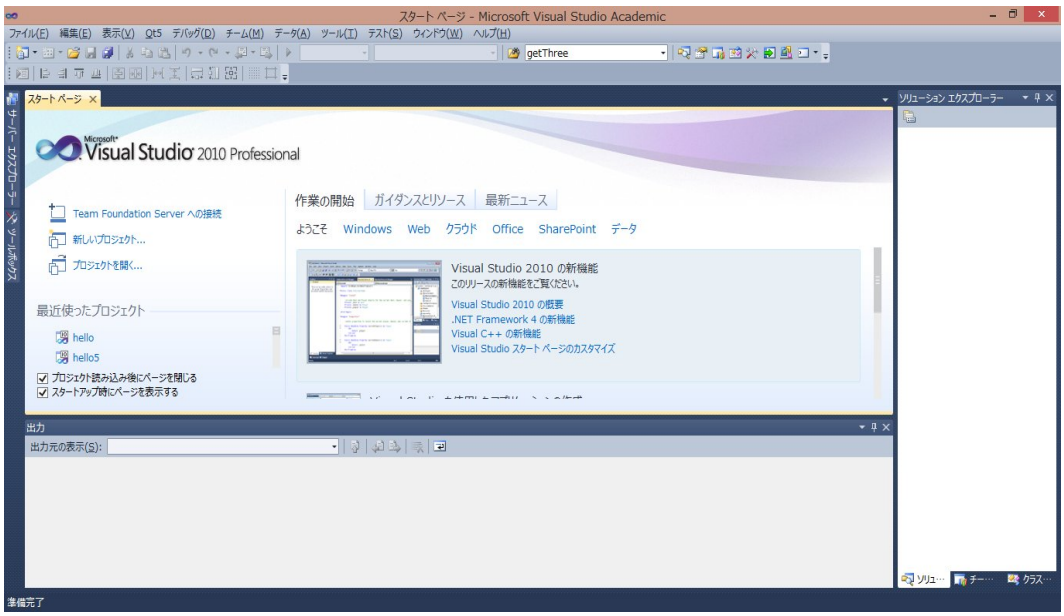
auto	double	int	struct	break	else	long	switch
case	enum	register	typedef	char	extern	return	union
const	float	short	unsigned	continue	for	signed	void
default	goto	sizeof	volatile	do	if	static	while
asm	**_cs**	**_ds**	**_es**	**_ss**	cdecl	far	huge
interrupt	near	pascal					

C++ では更に次の識別子もキーワードです。

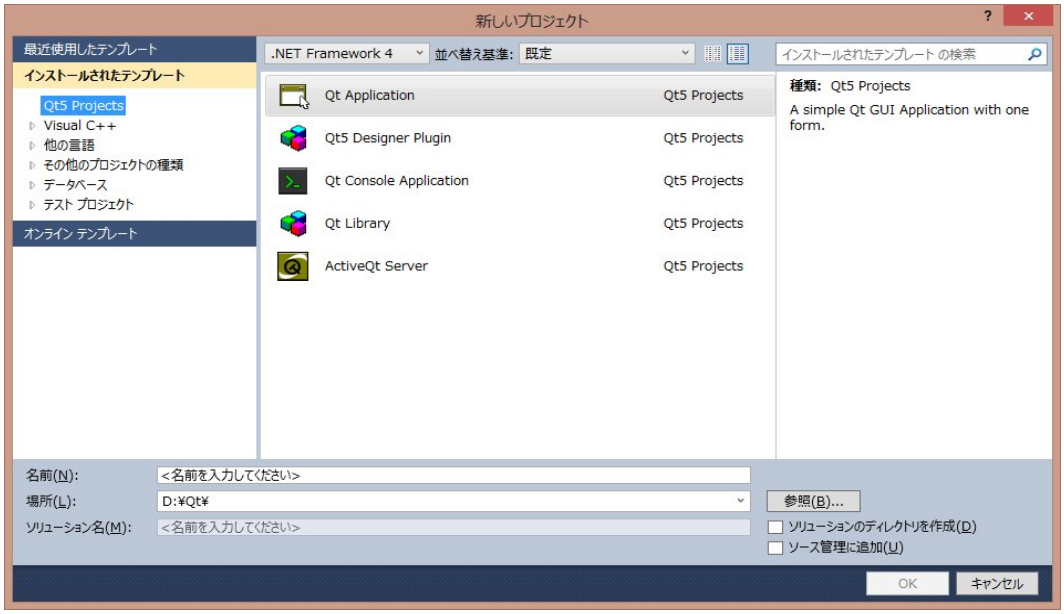
class	inline	overload	public	friend	new	virtual	operator
this	delete						

それではプログラミングを始めましょう。次は三つの数を入力し、その和と平均と標準偏差を出力する C++ のプログラムです。

まず、Microsoft Visual Studio 2010 を立ち上げる。

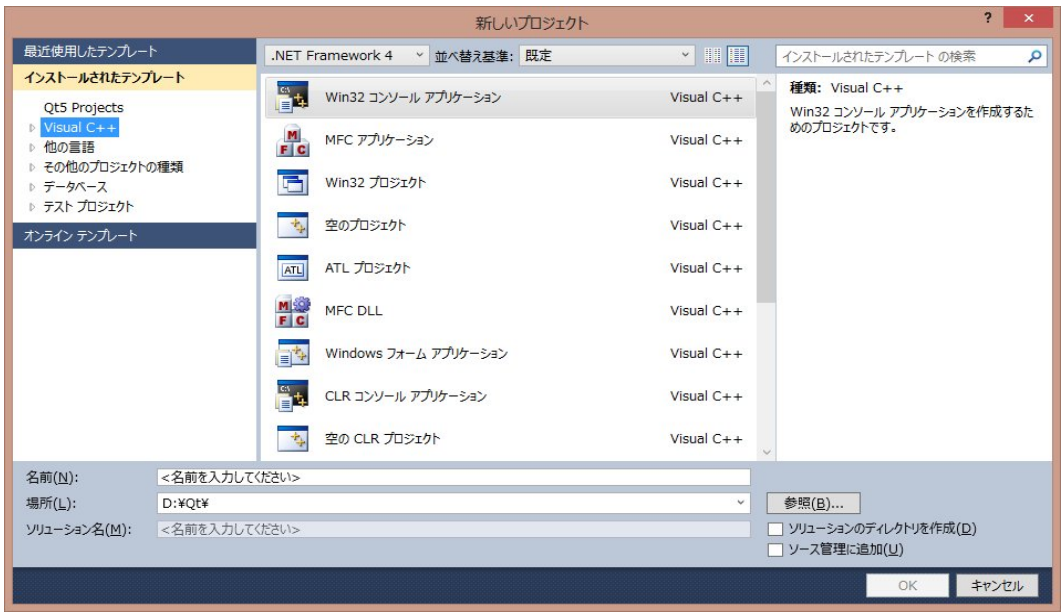


すでに立ち上げていれば、「ファイル」メニューの「閉じる」でプロジェクトを閉じます。そして、「表示」のメニューで「スタートページ」をクリックします。「スタートページ」で「新しいプロジェクト」をクリックするか、「ファイル」メニューの「新規作成」の「プロジェクト」を選択します。



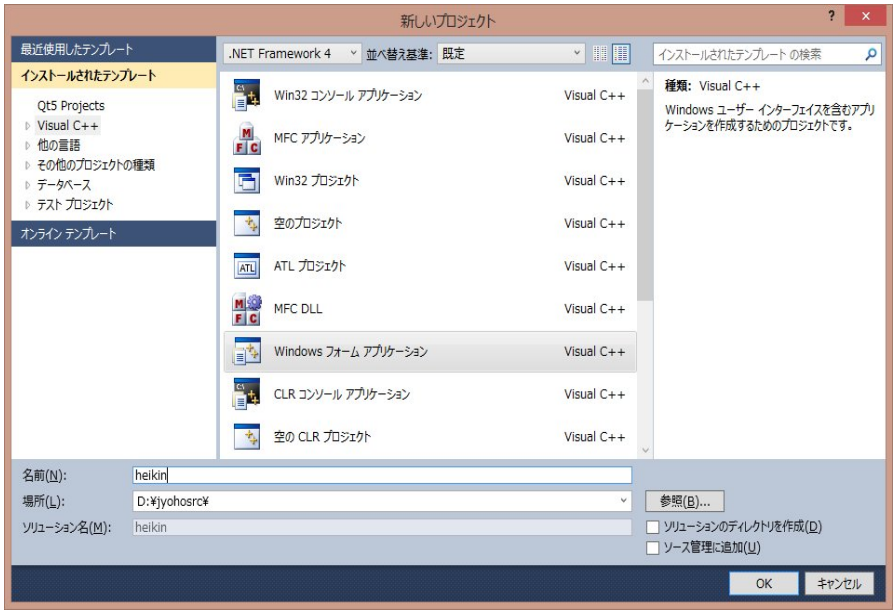
新しい q q qq プロジェクトのウインドウが開きます。私の現在の VC++ では、Qt5Project がデファルトで表示されていますが表示が同じでなくても気にしないで下さい。Qt は VC++ 固有の GUI 以外の別の GUI のプログラムを作るための無料のパッケジです。linux で GUI のプログラムを作る時の一つの選択肢です。

Visual C++ をクリック（選択）します。

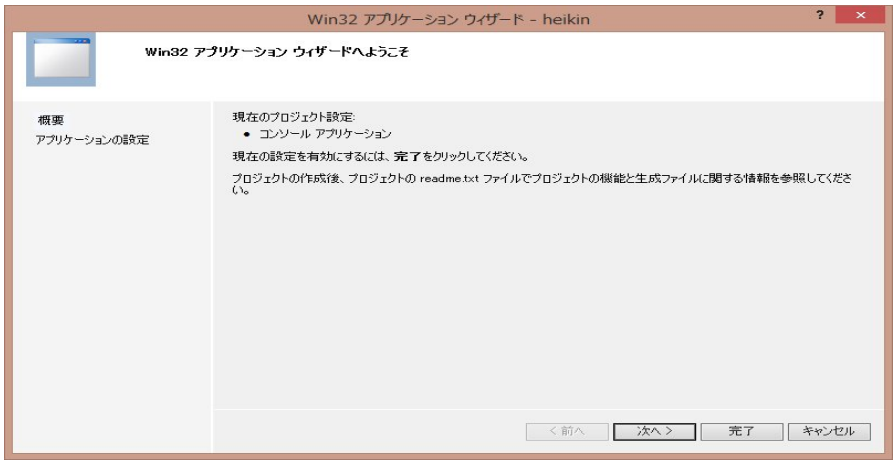


ここでも表示が違っていても気にしないで良いです。Win32 コンソールアプリケーションがあれば良いです。

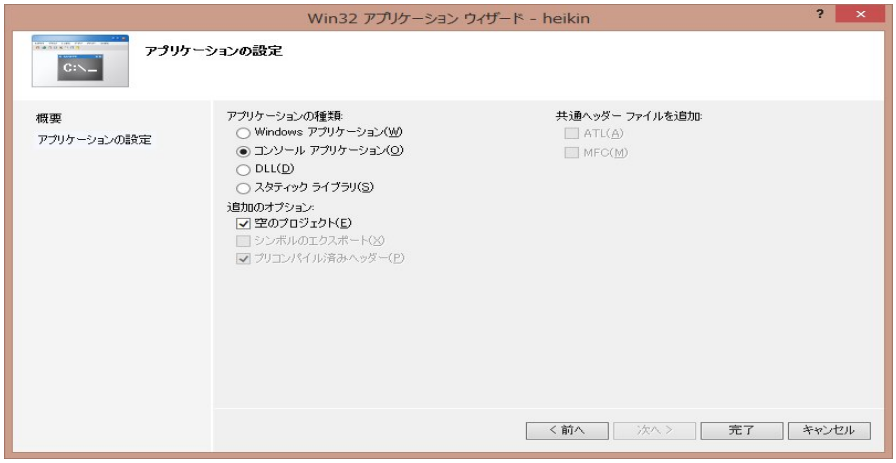
「Win32 コンソール　アプリケーション」をクリックし、図では場所 (L):が D:\Qt となっています（自分で適当なディレクトリ、例えば D:\jhohosrc\を作り、ここを指定すれば (「参照」のボタンをクリックし、適当なフォルダを選択すれば良いです)、プログラムが D:\jyuhosrc\に作られます）が、何処にプログラムが保存されるか気にしなければ、そのままで良いですので、名前 (N): のエディットボックスに例えば heikin と入力し、



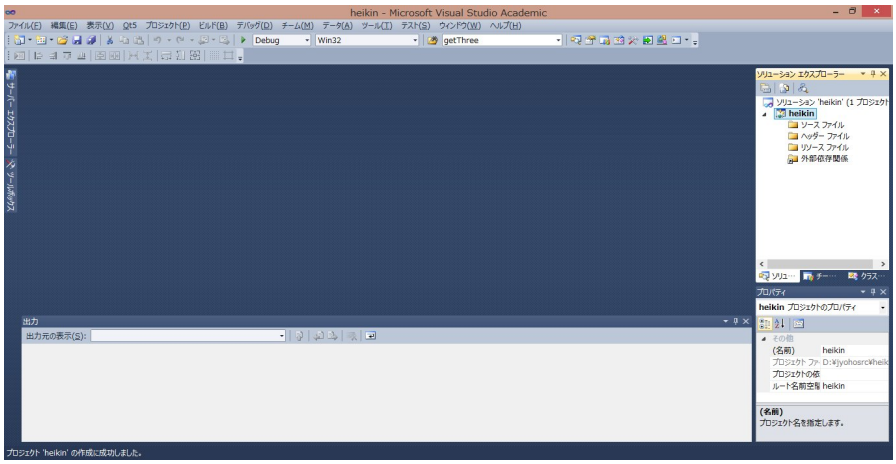
OK ボタンをクリックする。



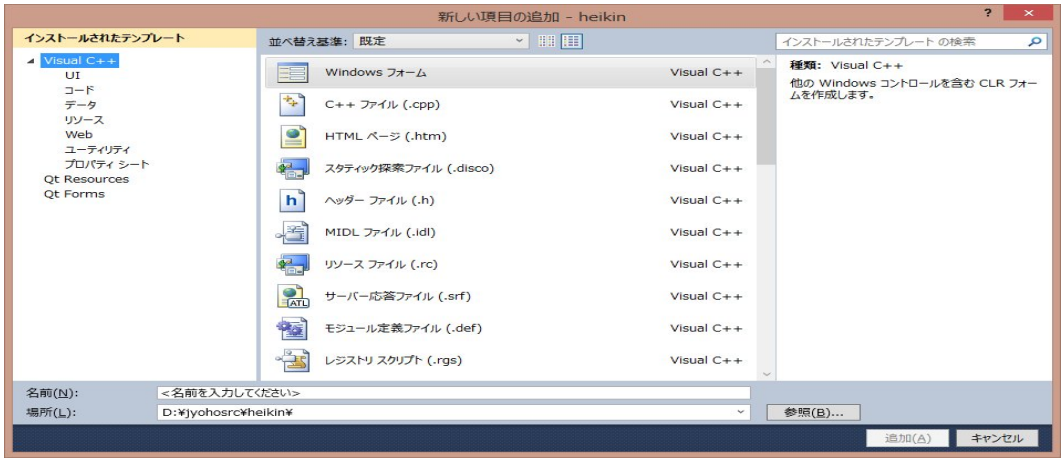
次へのボタンをクリックし、空のプロジェクトをチェックする。



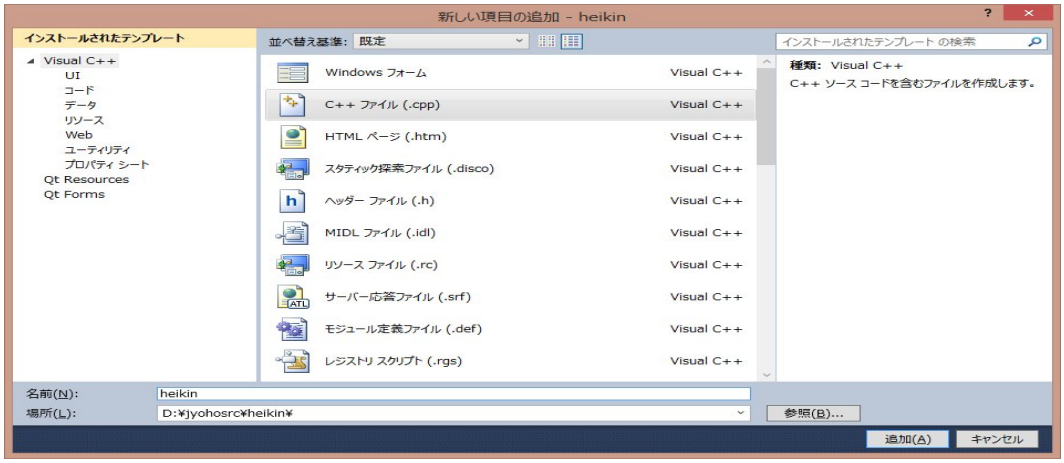
完了ボタンをクリックする。



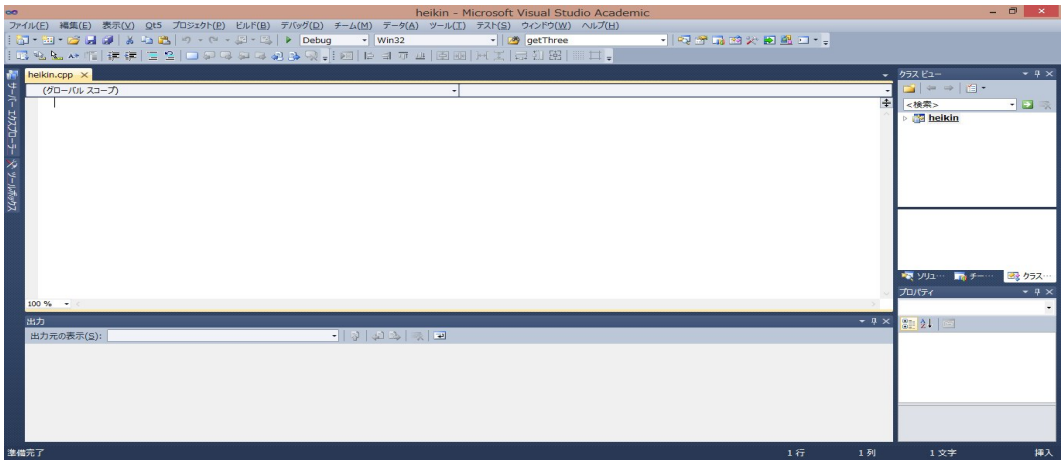
一段目の左から二つ目のアイコン（新しい項目の追加）をクリックする。



C++ ファイルをクリックし、名前の欄に heikin と打ち込む。このファイルの名前は何でも良いです。



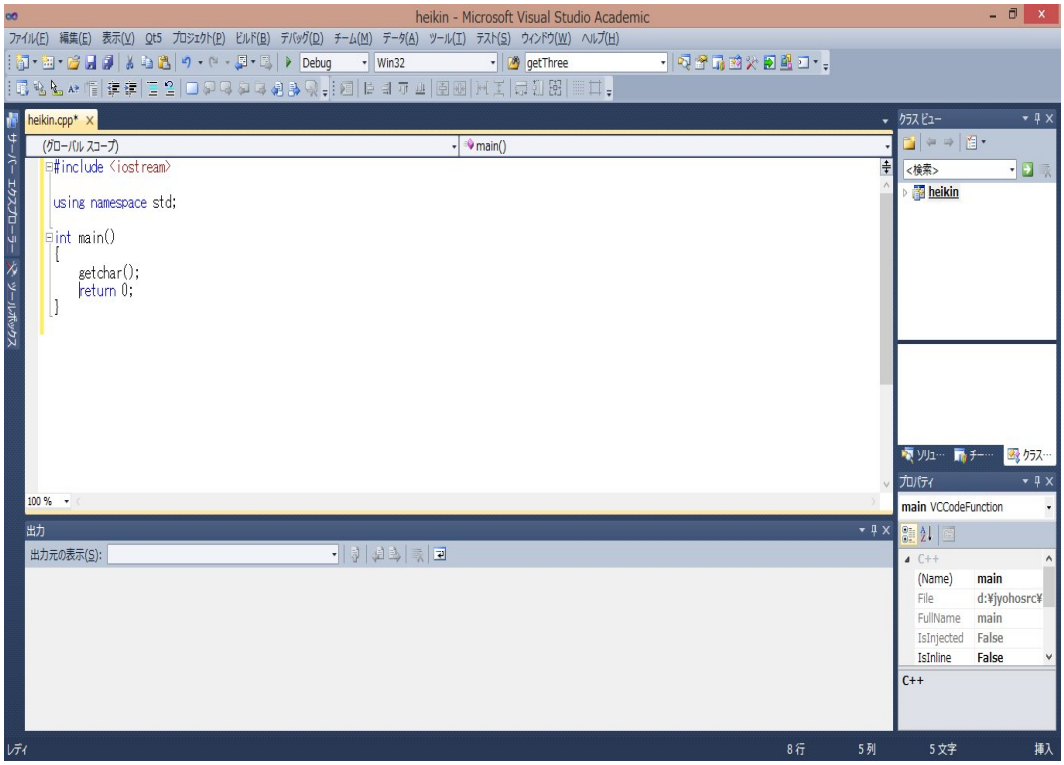
追加のボタンをクリックする。



heikin.cpp のファイルが作られ、開いている。ここにプログラムを打ち込みます。
　main() 関数の引数は使用しないので

```
#include <iostream>
using namespace std;

int main()
{
        getchar();
        return 0;
}
```

が雛形です。これから始めます。



三つの数を入力してもらうので、その数字を保存する変数が必要です。変数は名前とどのような値を保存するかをコンパイラに教える必要があります。名前だけでなくどのような値を保存するか指示しなければいけないのがインタプリッタ系の Python や Sheme と異なるところです。コンパイラでは順番に実行しながらプログラムを調べていくわけではなく、プログラム全体をアセンブリ言語そして機械語に翻訳してから実行するので、変数に何が入るか教えてやらないとどれだけの大きさのメモリが必要かコンパイラには分からないからです。C++ では数値を整数値と実数値に分けて考えます。整数値と実数値（小数点の付いた数）では保存方法が違います。必要とするメモリの大きさに応じて色々準備されていますが、この講義では整数値は int を、実数値は　double を使うことにします。名前は a と b と c を使うことにします。変数や関数は使う前に宣言しておかなくてはいけません。

```
#include <iostream>
using namespace std;

int main()
{
        double a, b, c;

        getchar();
        return 0;
}
```

とします。

三つの数を入力してもらうためには、三つの数を入力してほしいと伝えなければなりません。cout を使って次のようにします。

```
#include <iostream>
using namespace std;

int main()
{
        double a, b, c;

        cout << "a b c = ";
        getchar();
        return 0;
}
```

とします。キー入力を受け付けるには cin を使います。

```
#include <iostream>
using namespace std;

int main()
{
        double a, b, c;

        cout << "a b c = ";
        cin >> a >> b >> c;
        getchar();
        return 0;
}
```

とします。

```
        cin >> a >> b >> c;
```

で、スペースで区切られた三つの数値が実数値として、順番に a, b, c にセットされます。

和と平均と標準偏差を計算するので、それらを保存する変数が必要です。double の変数とし、wa, heikin, hyoujyunhensa という名前とします。d, e, f でも a2, b2, c2 でも良いですがなるべく分かりやすい名前にします。これを最初の変数の宣言に追加します。

```
#include <iostream>
using namespace std;

int main()
{
        double a, b, c, wa, heikin, hyoujyunhensa;
```

```
        cout << "a b c = ";
        cin >> a >> b >> c;
        getchar();
        return 0;
}
```

となりました。和は数学の時と同じく $a+b+c$ で表現されます。それを wa に保存するには

```
        wa = a + b + c;
```

で表現します。代入文と言います。文や宣言の終わりにはセミコロン ; が必要なことを忘れないようにしてください。

```
#include <iostream>
using namespace std;

int main()
{
        double a, b, c, wa, heikin, hyoujyunhensa;

        cout << "a b c = ";
        cin >> a >> b >> c;
        wa = a + b + c;
        getchar();
        return 0;
}
```

となりました。

平均は和を 3 で割れば良いです。割り算は / で表現します。$wa/3$ を wa に代入します。

```
      heikin = wa / 3;
```

を追加します。

```
#include <iostream>
using namespace std;

int main()
{
        double a, b, c, wa, heikin, hyoujyunhensa;

        cout << "a b c = ";
        cin >> a >> b >> c;
        wa = a + b + c;
        heikin = wa / 3;
        getchar();
        return 0;
}
```

となりました。最後は標準偏差の計算です。標準偏差は分散の平方根で与えられます。分散は平均との差の平方の和

$$(a-heikin)\times(a-heikin)+(b-heikin)\times(b-heikin)+(c-heikin)\times(c-heikin)$$

を 3 で割ったものですから

$$((a-heikin)\times(a-heikin)+(b-heikin)\times(b-heikin)+(c-heikin)\times(c-heikin))/3$$

となりますが、C++ では × は * で表します。平方根は数学関数 sqrt() で与えられます。これを使うためには sqrt() 関数の宣言が必要です。それはコンパイラのメーカーが準備してくれていて、math.h という名前のファイルで宣言しています。 sqrt() や sin() や cos() と言いた関数を使うためには math.h を読み込む必要があります。それを実行するのが include 文です。

```
#include <math.h>
```

を追加します。

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
        double a, b, c, wa, heikin, hyoujyunhensa;

        cout << "a b c = ";
        cin >> a >> b >> c;
        wa = a + b + c;
        heikin = wa / 3;
        getchar();
        return 0;
}
```

となりました。標準偏差のための代入文

```
        hyoujyunhensa =
            sqrt(((a-heikin)*(a-heikin)+(b-heikin)*(b-heikin)+(c-heikin)*(c-heikin))/3);
```

を追加します。式が長ければ数行に渡って書いても良いです。

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
```

```
        double a, b, c, wa, heikin, hyoujyunhensa;

        cout << "a b c = ";
        cin >> a >> b >> c;
        wa = a + b + c;
        heikin = wa / 3;
        hyoujyunhensa =
            sqrt(((a-heikin)*(a-heikin)+(b-heikin)*(b-heikin)+(c-heikin)*(c-heikin))/3);
        getchar();
        return 0;
}
```

となりました。必要な計算が出来たので、結果を表示します。

```
        cout << "和  = " << wa << endl;
```

で、「和　＝」という文字列と wa に代入されている実数値が表示され、改行されます。

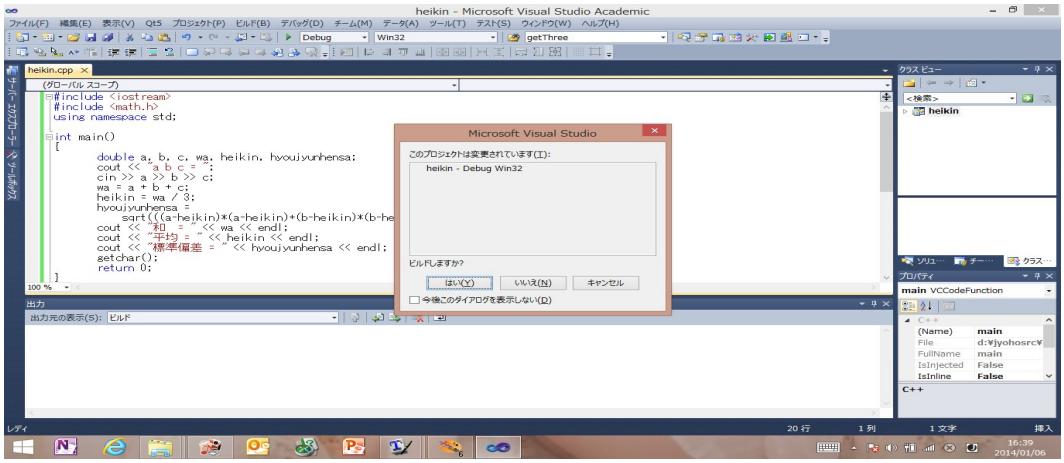
```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
        double a, b, c, wa, heikin, hyoujyunhensa;

        cout << "a b c = ";
        cin >> a >> b >> c;
        wa = a + b + c;
        heikin = wa / 3;
        hyoujyunhensa =
            sqrt(((a-heikin)*(a-heikin)+(b-heikin)*(b-heikin)+(c-heikin)*(c-heikin))/3);
        cout << "和  = " << wa << endl;
        getchar();
        return 0;
}
```

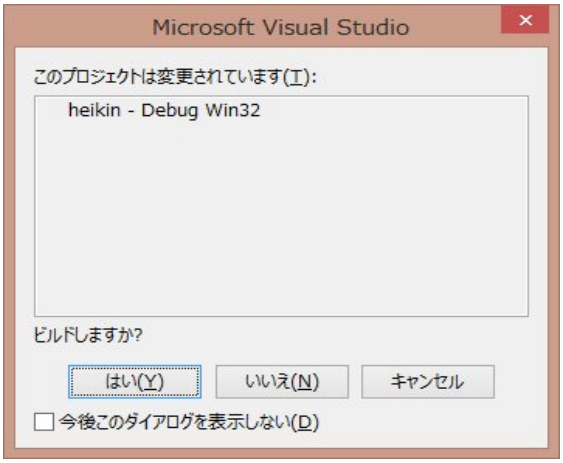
となりました。 同様に平均と標準偏差を表示します。

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
```

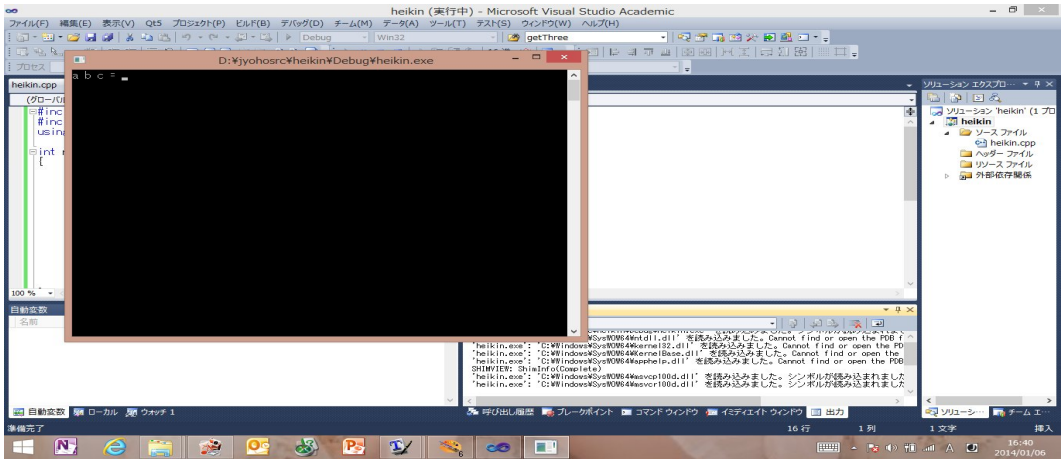
```
        double a, b, c, wa, heikin, hyoujyunhensa;
        cout << "a b c = ";
        cin >> a >> b >> c;
        wa = a + b + c;
        heikin = wa / 3;
        hyoujyunhensa =
            sqrt(((a-heikin)*(a-heikin)+(b-heikin)*(b-heikin)+(c-heikin)*(c-heikin))/3);
        cout << "和  = " << wa << endl;
        cout << "平均 = " << heikin << endl;
        cout << "標準偏差 = " << hyoujyunhensa << endl;
        getchar();
         return 0;
}
```

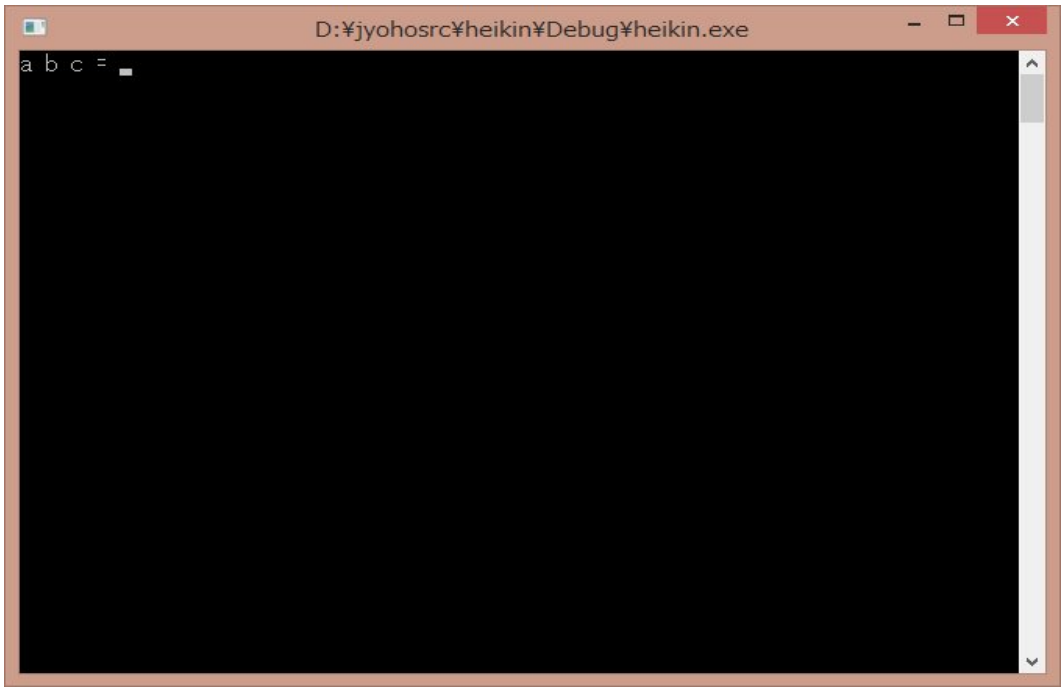
となります。これでプログラムが出来ました。コンパイルし実行します。Debug と書いてある処の左の三角をクリックするか、「デバッグ」メニューの「デバッグ開始」を選択します。





「はい」のボタンをクリックします。





黒い画面が現れました。a b c = と表示されています。例えば、 1 2 3 と入力してみます。一瞬、何かが表示され画面が消えてしまいます。

```
getchar();
```

で、キー入力を待機するはずなのに、何が起こったのでしょうか？これは

```
cin >> a >> b >> c;
```

で、数値を入力する時、最後に Enter キーを押します。この Enter キーが入力されたという情報が消費されずに残っていて

```
getchar();
```

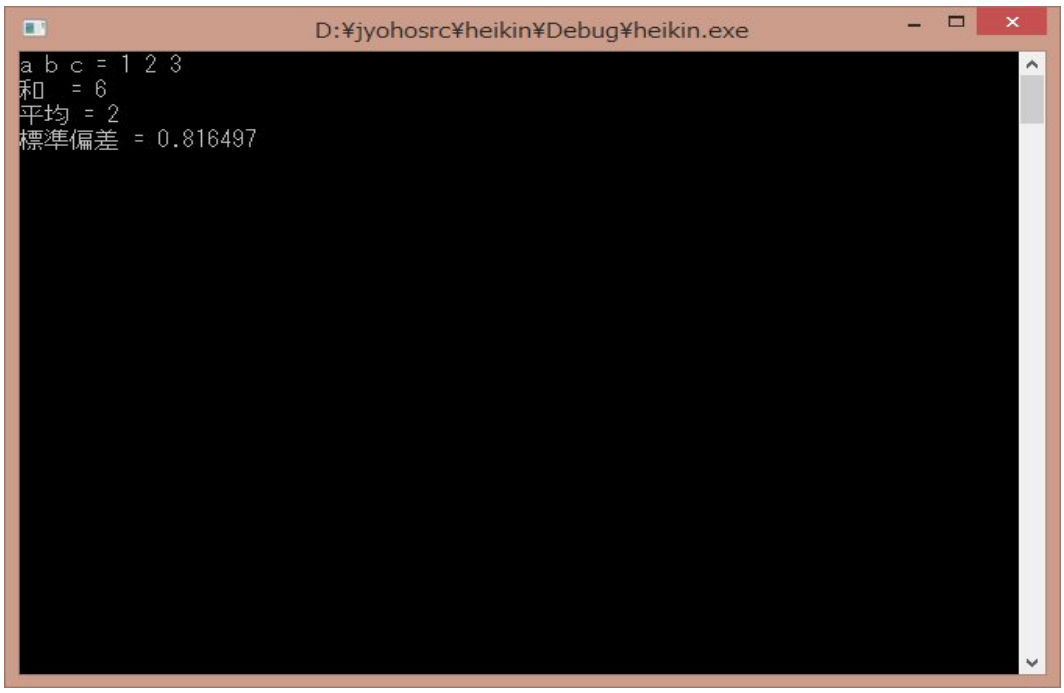
でそれを使い、すべての命令が実行されたので、プログラムは終了し、画面が消えました。画面が消えなくするには、もう一個

```
        getchar();
```

を追加しておけば良かったです。

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
        double a, b, c, wa, heikin, hyoujyunhensa;
        cout << "a b c = ";
        cin >> a >> b >> c;
        wa = a + b + c;
        heikin = wa / 3;
        hyoujyunhensa =
            sqrt(((a-heikin)*(a-heikin)+(b-heikin)*(b-heikin)+(c-heikin)*(c-heikin))/3);
        cout << "和  = " << wa << endl;
        cout << "平均 = " << heikin << endl;
        cout << "標準偏差 = " << hyoujyunhensa << endl;
        getchar();
        getchar();
        return 0;
}
```

でプログラムは完成です。

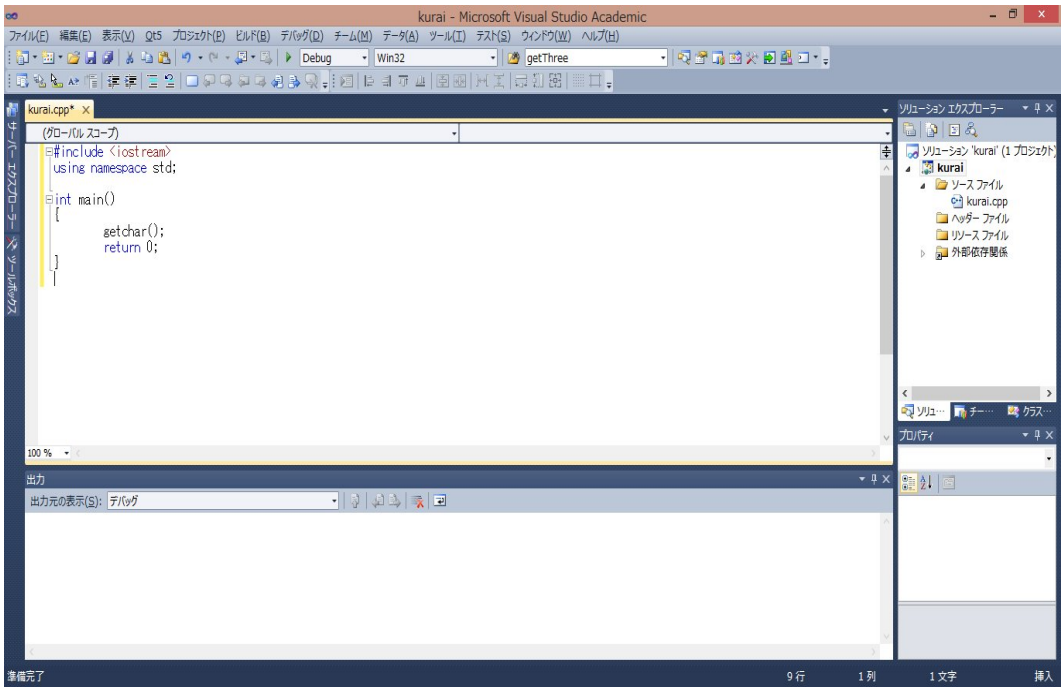


次に三桁の整数を入力して百、十、一の位の数を表示するプログラムを作ります。

上と同様にして、例えば、kurai という名前の空の Win32 コンソールアプリケーションのプロジェクトを作り、kurai.cpp という C++ のファイルを追加します。雛形

```
#include <iostream>
using namespace std;

int main()
{
        getchar();
        return 0;
}
```

を打ち込みます。



三桁の整数を入力するので、入力された数値を保存する変数を準備します。整数値なので int の変数です。名前は n とします。大した意味は有りませんが、Fortran の伝統で、整数値の変数は、i や j や k や m や n を使うことが多いです。

int は 2Byte で -32768 から 32767 までの整数値を記憶する事ができます。これの値はコンパイラに依ります。4Byte で -2147483648 から 2147483647 までの整数値であることもあります。もっと大きい整数値を記憶したければ long int を使います。long int は 4Byte で -2147483648 から 2147483647 までの整数値を記憶できます。これより大きな整数値を使いたければ、後で出てくる配列を使います。

三桁の整数を入力して百、十、一の位の数を表示するので、それぞれのくらいの数字を記憶する変数も必要です。これらを int の変数で、名前をそれぞれ hyaku, juu, iti とします。

変数の宣言

```
        int n, hyaku, juu, iti;
```

を追加します。

```
#include <iostream>
using namespace std;

int main()
{
        int n, hyaku, juu, iti;

         getchar();
         return 0;
}
```

となります。三桁の整数を入力します。

```
        cout << "三桁の数 ";
        cin >> n;
```

です。これで n に入力された整数値が代入されます。プログラムは

```
#include <iostream>
using namespace std;

int main()
{
        int n, hyaku, juu, iti;

         cout << "三桁の数 ";
         cin >> n;
         getchar();
         return 0;
}
```

となります。
　三桁の整数の百の位は 100 で割った商です。これは

```
        hyaku = n / 100;
```

で計算できます。C++ の整数同士の割り算は割り切れなければ小数点以下が切り捨てられます。
　プログラムは

```
#include <iostream>
using namespace std;

int main()
```

```
{
        int n, hyaku, juu, iti;

         cout << "三桁の数 ";
         cin >> n;
         hyaku = n / 100;
         getchar();
         return 0;
}
```

となります。三桁の整数の十の位を求めるために、もう元の三桁の整数は使いませんから、三桁の整数の百の位を除いた数を n に入れることにします。やり方は複数考えられますがここでは三桁の整数を 100 で割った余りの数を求めることとします。余りを求める演算子は Python の時出てきた % です。n を 100 で割った余りを n に保存するには

```
        n = n % 100;
```

とします。＝　は数学の等号と異なり、プログラミングでは代入を意味します。これはまた簡潔に

```
        n %= 100;
```

と表現することも出来ます。n を 100 で割った余りを n とするという意味です。これを追加するとプログラムは

```
#include <iostream>
using namespace std;

int main()
{
        int n, hyaku, juu, iti;

         cout << "三桁の数 ";
         cin >> n;
         hyaku = n / 100;
         n %= 100;
         getchar();
         return 0;
}
```

となります。n には二桁（または一桁）の数が代入されているので、十の位を求めるために 10 で割った商ですから、上と同様

```
        juu = n / 10;
```

で計算できます。これを追加するとプログラムは

```
#include <iostream>
using namespace std;

int main()
{
        int n, hyaku, juu, iti;

         cout << "三桁の数 ";
         cin >> n;
         hyaku = n / 100;
         n %= 100;
         juu = n / 10;
         getchar();
         return 0;
}
```

となります。n の一の位は n を 10 で割った余りですから

```
        iti = n % 10;
```

で計算できます。これを追加するとプログラムは

```
#include <iostream>
using namespace std;

int main()
{
        int n, hyaku, juu, iti;

         cout << "三桁の数 ";
         cin >> n;
         hyaku = n / 100;
         n %= 100;
         juu = n / 10;
         iti = n % 10;
         getchar();
         return 0;
}
```

となります。次は計算結果の表示です。

```
        cout << "百の位 = " << hyaku << endl;
        cout << "十の位 = " << juu << endl;
        cout << "一の位 = " << iti << endl;
```

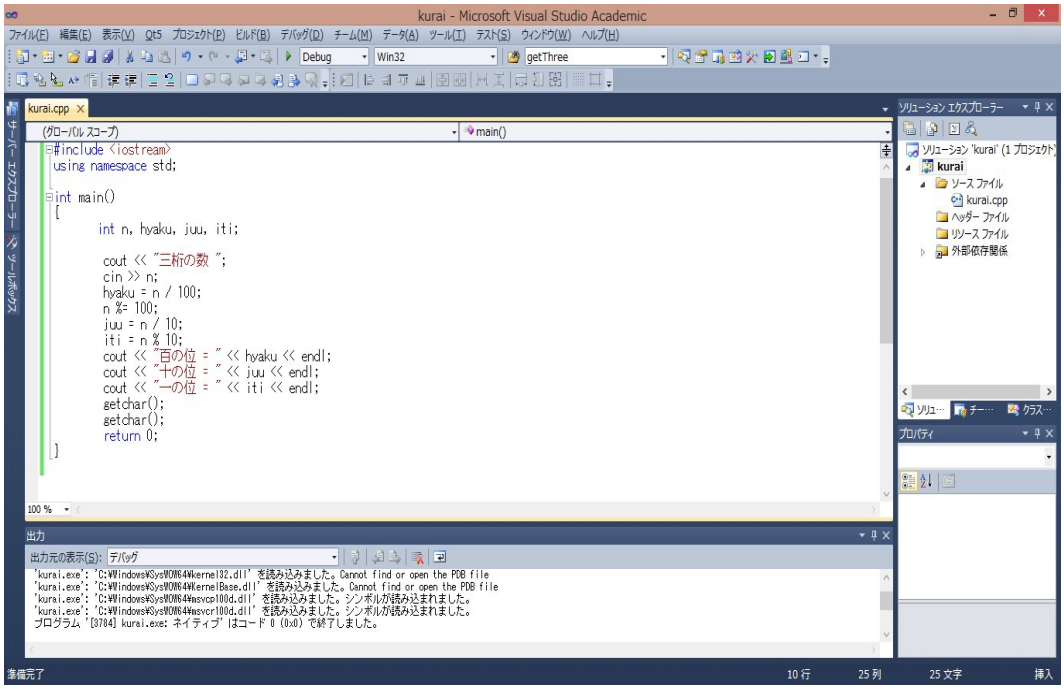
で良いです。これを追加するとプログラムは

```
#include <iostream>
using namespace std;

int main()
{
        int n, hyaku, juu, iti;

         cout << "三桁の数 ";
         cin >> n;
         hyaku = n / 100;
         n %= 100;
         juu = n / 10;
         iti = n % 10;
         cout << "百の位 = " << hyaku << endl;
         cout << "十の位 = " << juu << endl;
         cout << "一の位 = " << iti << endl;
         getchar();
         return 0;
}
```

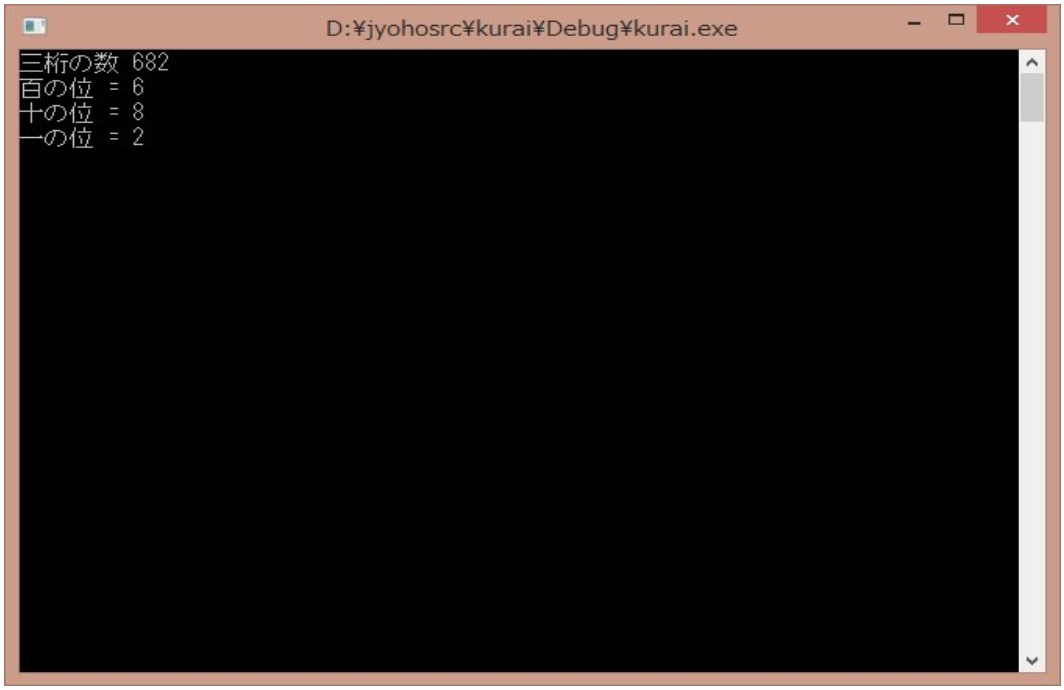
となります。最後に、cin を使ったので、getchar(); をもう一つ追加しておきます。プログラムは

```
#include <iostream>
using namespace std;

int main()
{
        int n, hyaku, juu, iti;

         cout << "三桁の数 ";
         cin >> n;
         hyaku = n / 100;
         n %= 100;
         juu = n / 10;
         iti = n % 10;
         cout << "百の位 = " << hyaku << endl;
         cout << "十の位 = " << juu << endl;
         cout << "一の位 = " << iti << endl;
         getchar();
         getchar();
         return 0;
}
```

となります。これでプログラムは完成です。



実行してみると



のようになります。

次のプログラムは atan() 関数を用いて円周率をもとめる C のプログラムである。

```
#include <stdio.h>
#include <math.h>
void main()
```

```
{
    printf("円周率=%20.15f\n", 4 * atan(1.0));
    getchar();
}
```

ここでは

```
void main()
{
}
```

と main() 関数の戻り値を void （値を戻さない）と宣言しているが、このようにしても良いです。この場合は return 文は必要ありません。

C++ はほぼ C を含んでいるのでこれを C++ のプログラムであるといっても間違いではありません。今までと入出力のやり方が違っています。複雑な書式の指定が必要でその説明だけでも時間がもったいないのでこの講義では説明はしません。ただ眺めるだけにして下さい。

atan() は倍精度浮動小数点数の引数を取り、その arctangent を倍精度浮動小数点数として返す関数である。数学関数を用いるためには

```
#include <math.h>
```

のインクルード文が必要である。以下にあげるのはよく使われる数学関数である。いずれも一つまたは二つの double の引数をとり、double の値を返す。

sin(x)	x の sine, 単位はラジアン
cos(x)	x の cosine, 単位はラジアン
tan(x)	x の tangent, 単位はラジアン
exp(x)	指数関数 e^x
log(x)	x の自然対数（基数は e）($x > 0$)
log10(x)	x の常用対数（基数は１０）($x > 0$)
pow(x, y)	x^y
sqrt(x)	x の平方根（$x \geq 0$)
fabs(x)	x の絶対値
floor(x)	x を越えない最大の整数
ceil(x)	x 以上の最小の整数
asin(x)	x の arcsine, 返される値の単位はラジアン
acos(x)	x の arccosine, 返される値の単位はラジアン
atan(x)	x の arctangent, 返される値の単位はラジアン

printf() 関数の書式の `%20.15f` は double の値を２０桁取り小数点以下１５桁表示せよとの指示である。これを C++ でやろうとすると厄介です。

C++ のプログラムは次のようになります。

```
#include <iostream>
#include <math.h>
using namespace std;

void main()
```

```
{
    cout << "円周率=" << 4 * atan(1) << endl;
    getchar();
}
```

C++ で、値を２０桁取り小数点以下１５桁表示せよと言った指示はめんどくさいのでここでは示さない。気になるなら、自分でインターネットで調べなさい。

もう、以下の例題のプログラムは一行ずつ解説しなくても理解できるはずです。

例題：球の半径 r の値を入力し、表面積と体積を求める C++ のプログラムを作れ。

解答例　次のようなプログラムを作ればいいです。

```
#include <iostream>
#include <stdio.h>
#include <math.h>

using namespace std;

int main(int argc, char* argv[])
{
    double PI = 4 * atan(1.0);
    double r, V, S;

    cout << "半径：";
    cin >> r;
    S = 4 * PI * r * r;
    V = 4 * PI * r * r* r / 3;
    cout << "S=" << S << " V=" << V << endl;
    getchar();
    getchar();
    return 0;
}
```

この例題では

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

という include 文が入ってますが、これは

```
    getchar();
```

の宣言がファイル stdio.h でなされているからです。しかし、上でやっていたように

```
#include <iostream>
```

があれば、getchar() の宣言もやってくれるので必要はないですが、あってもエラーにはなりません。

例題：標準体重は 身長 (m)× 身長 (m) ×22 の ±10% 以内です。身長を入力し、その標準体重を出力するプログラムを作れ。

```
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
    double A, W, L, H;

    cout << "身長 (m)=";
    cin >> A;
    getchar();
    W = A * A * 22;
    L = W * 0.9;
    H = W * 1.1;
    cout << "標準体重（kg）は" << L << "から" << H << "の範囲です" << endl;
    getchar();
    return 0;
}
```

例題： 2^4 を、 $2*2*2*2$ ではなく、底が **e** の対数関数 log() と底が **e** の指数関数 exp() を利用して、計算するプログラムを作れ。

```
#include <iostream>
#include <math.h>

using namespace std;

int main(int argc, char* argv[])
{
    double a, b;

    cout << "底＝";
    cin >> a;
    cout << "指数＝";
    cin >> b;
    getchar();
    cout << a << "の" << b << "乗は" << exp(b*log(a)) << "です" << endl;
    getchar();
    return 0;
}
```

例題： 余弦定理 $c^2 = a^2 + b^2 - 2ab\cos C$ を利用して、**a,b,c** の値から三角形の角 **C** を求めるプログラムを作れ。

```
#include <iostream>
```

```
#include <math.h>

using namespace std;

int main(int argc, char* argv[])
{
    double a, b, c, C;
    double PI = 4*atan(1.0);

    cout << "a="; cin >> a;
    cout << "b="; cin >> b;
    cout << "c="; cin >> c;
    getchar();
    C = acos((a*a+b*b-c*c)/(2*a*b))/PI*180;
    cout << "C=" << C << endl;
    getchar();
    return 0;
}
```

演習問題

1. 自然数 n を入力し n を 3 で割ったときの商と余りを表示するプログラムを作れ。
2. ４桁の数を入力して千、百、十、一の位の数を出力するプログラムを書け。
3. 対数関数 LOG() または log10(x) を利用して入力した数の桁数を出力するプログラムを書け。
4. 余弦定理 $c^2 = a^2 + b^2 - 2ab\cos C$ を利用して、a,b,C の値から三角形の辺 AB の長さ c を求めるプログラムを作れ。
5. 三角形 ABC の辺 BC の長さ a と角 A から、外接円の半径 R を求めるプログラムを作れ

旺文社の昔の数学 A の教科書には、次のような最大公約数を求めるプログラムが載っています。

```
10 REM 最大公約数
20 INPUT "N, M"; N, M
30 G=N
40 IF N>INT(N/G)*G THEN 80
50 IF M>INT(M/G)*G THEN 80
60 PRINT "最大公約数＝";G
70 GOTO 100
80 G=G-1
90 GOTO 40
100 END
```

このプログラムの説明はしません。気になるなら私の Java による BASIC インタプリタのホームページをみて、その解説やサンプルプログラムを読んでください。次の C++ による直訳と見比べると大体の感じは分かると思います。BASIC は死んだ言語です。

これを C++ に翻訳すると次のようになります。

```
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
    int n, m, g;

    cout << "n m:";
    cin >> n >> m;
    getchar();
    g = n;
L0: if (n % g != 0) goto L1;
    if (m % g != 0) goto L1;
    cout << "最大公約数＝" << g << endl;
    goto L2;
L1: g = g-1;
    goto L0;
L2: getchar();
    return 0;
}
```

これは昔のプログラミングスタイルで、読みにくいプログラムになります。

```
    goto L2;
```

は goto 文と言われ、この場合は L2 というラベルの付いた文に制御を移します。つまり、

```
L2: getchar();
```

を実行します。ラベルは文字列にコロン : を付けて表します。BASIC の場合は行番号で分岐先を指示しています。

```
40 IF N>INT(N/G)*G THEN 80
```

は、N が G で割り切れないなら８０行にジャンプしなさいという IF 文です。

このプログラムのように goto 文を多用すると「スパゲッチイプログラム」という読みにくいプログラムになるので、 PASCAL や C と言った構造的プログラミング言語が作られました。n と m をスペースで区切って、続けて入力します。

これらプログラムのアルゴリズムは

1. N を入力する
2. M を入力する
3. G を N から N-1, N-2, N-3, ···, 1 と、G が N と M も共に割り切るまで、一つずつ減らしていく。G が N と M も共に割り切るとき、G は N と M の最大公約数である。

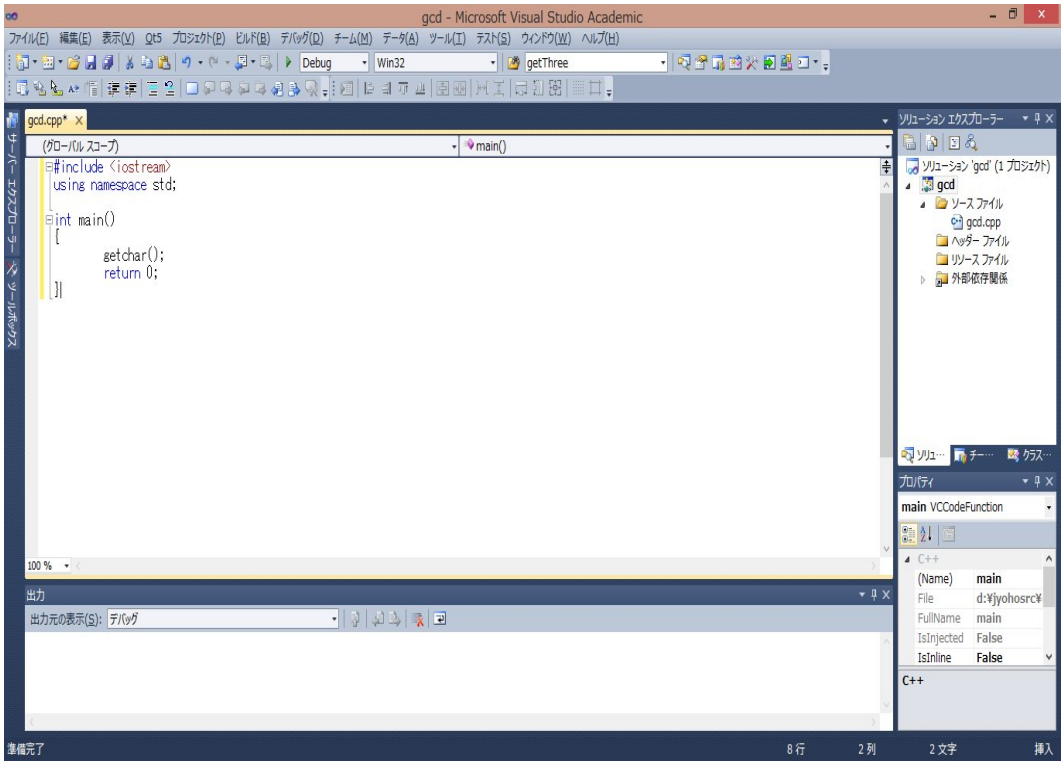
4. G を N と M の最大公約数として表示する。

です。

このアルゴリズムを元にプログラムを作ってみます。今まで通り雛形

```
#include <iostream>
using namespace std;

int main()
{
        getchar();
        return 0;
}
```

から始めます。



入力される二つの整数の為の変数と最大公約数を入れる為の変数の 3 個が少なくとも必要です。これらは int の変数で、名前は n, m, g とします。まず、変数の宣言です。

```
        int n, m, g;
```

を追加します。

```
#include <iostream>
using namespace std;
```

```
int main()
{
        int n, m, g;

        getchar();
        return 0;
}
```

となります。まず、二つの整数を入力します。

```
        cout << "n m:";
        cin >> n >> m;
```

を追加します。

```
#include <iostream>
using namespace std;

int main()
{
        int n, m, g;

        cout << "n m:";
        cin >> n >> m;
        getchar();
        return 0;
}
```

となります。スペースで区切られた二つの整数が n と m にセットされました。
g を n から n-1, n-2, n-3, ⋯, 1 と、 g が n と m も共に割り切るまで、一つずつ減らしていく。
という部分をプログラミングする必要があります。
g を n から n-1, n-2, n-3, ⋯, 1 と、一つずつ減らしていく。
というような繰り返しの場合、C++ では、 for 文か while 文か do while 文を使いますが、このように繰り返しの規則がはっきりしている場合は普通 for 文を使います
今の場合、次の形の for 文を使えば良いです。

```
       for (g = n; g > 0; g--) {
               ................
        }
```

for 文は

```
for (初期設定; 条件式; 後処理) {
    命令の並び
}
```

の形式で、まず、初期設定を実行し、条件式が真であれば、命令の並びを実行し、後処理をし、条件式が真であれば、命令の並びを実行し、後処理をし、を条件式が偽になるまで繰り返します。条件式の一回目の判定が偽であれば、命令の並びは一回も実行されません。

```
for (g = n; g > 0; g--) {
        ..............
 }
```

の場合は、まず g に n を代入し（g の値を n に保存されている値と同じにする)、g が正であれば（g > 0 は数学の記号と同じ意味です)、{ と } の間に書かれている命令を実行し、g をひとつ減らし（g– です。これは　g = g - 1 と同じ意味で、C++ ではこのような簡潔な表現が出来ます。更に、これを前に n = n % 100 を n %= 100 と表現したように、 g -= 1 と表現することも出来ます)、再び、真ん中の条件式 g > 0 を調べます。g が正であれば、{ と } の間に書かれている命令を実行し、g をひとつ減らし、を繰り返し、g = n, n-1, n-2, n-3, ···, 1 と g の値を変えながら { と } の間に書かれている命令を実行します。

```
for (g = n; g > 0; g--) {
}
```

を追加します。

```
#include <iostream>
using namespace std;

int main()
{
        int n, m, g;

        cout << "n m:";
        cin >> n >> m;
        for (g = n; g > 0; g--) {
        }
        getchar();
        return 0;
}
```

となります。

g を n から n-1, n-2, n-3, ···, 1 と、 g が n と m も共に割り切るまで、一つずつ減らしていく。

の

g が n と m も共に割り切るまで、

の部分を for 文の{ と } の間に書きます。条件を判断するには if 文を使います。

if 文は、 Scheme の場合と異なり、 2 つの形式があります。

```
if (条件式) {
    命令の並び
}
```

の形で、条件式が真 (true) の時、命令の並びを実行するか

```
if（条件式）{
    命令の並び
} else {
    命令の並び
}
```

の形で、条件式が真 (true) の時、上の命令の並びを実行し、そうでないとき下の命令の並びを実行します。

```
if（条件式）{
    命令の並び
}
```

や

```
if（条件式）{
    命令の並び
} else {
    命令の並び
}
```

で、命令の並びが一個の命令の場合は、

```
if (i % 2 == 0)
    s += pow(x, 2*i) / fact(2*i);
else
    s -= pow(x, 2*i) / fact(2*i);
```

のように、命令の並びを囲む { と } は省略できます。

g が n を割り切るは条件式 `n % g == 0` で表現できます。 `n % g` は n を g で割った余りを与え、それが 0 と等しいかを判定しています。> や < が大小関係を判定する比較演算子であったように、== は等しいかどうかを判定する比較演算子です。比較演算子は算術演算子より優先順位が低いので括弧は必要ありません。勿論、`(n % g) == 0` と書いても良いです。同様に、g が m を割り切るは条件式 `m % g == 0` で表現できます。これが同時に成り立つことを問題にしているので論理演算子「かつ」で連結すればいいです。「かつ」を表す論理演算子は `&&` で表現します。論理演算子は比較演算子より優先順位が低いので括弧は必要ありません。従って、`n % g == 0 && m % g == 0` で g が n と m も共に割り切るかどうか判定できます。勿論、`(n % g == 0) && (m % g == 0)` としても良いです。

従って、今の場合、if 文は

```
if (n % g == 0 && m % g == 0) {
}
```

となります。

g を n から n-1, n-2, n-3, ⋯, 1 と、 g が n と m も共に割り切るまで、一つずつ減らしていく。

ですから、条件式 n % g == 0 && m % g == 0 が成り立てば、g を減らすのをやめますから、for 文から抜け出す必要があります。この為の命令が break 文です。
　if 文は

```
if (n % g == 0 && m % g == 0) {
        break;
}
```

となります。これをプログラムに追加します。

```
#include <iostream>
using namespace std;

int main()
{
        int n, m, g;

        cout << "n m:";
        cin >> n >> m;
        for (g = n; g > 0; g--) {
                if (n % g == 0 && m % g == 0) {
                        break;
                }
        }
        getchar();
        return 0;
}
```

となります。 break 文を実行したとき、g には n と m の最大公約数がセットされていますから。アルゴリズムの最後の行
　g を n と m の最大公約数として表示する。
を実行します。

```
        cout << "最大公約数=" << g << endl;
```

を追加します。

```
#include <iostream>
using namespace std;

int main()
{
        int n, m, g;

        cout << "n m:";
        cin >> n >> m;
```

```
        for (g = n; g > 0; g--) {
                if (n % g == 0 && m % g == 0) {
                        break;
                }
        }
        cout << "最大公約数=" << g << endl;
        getchar();
        return 0;
}
```

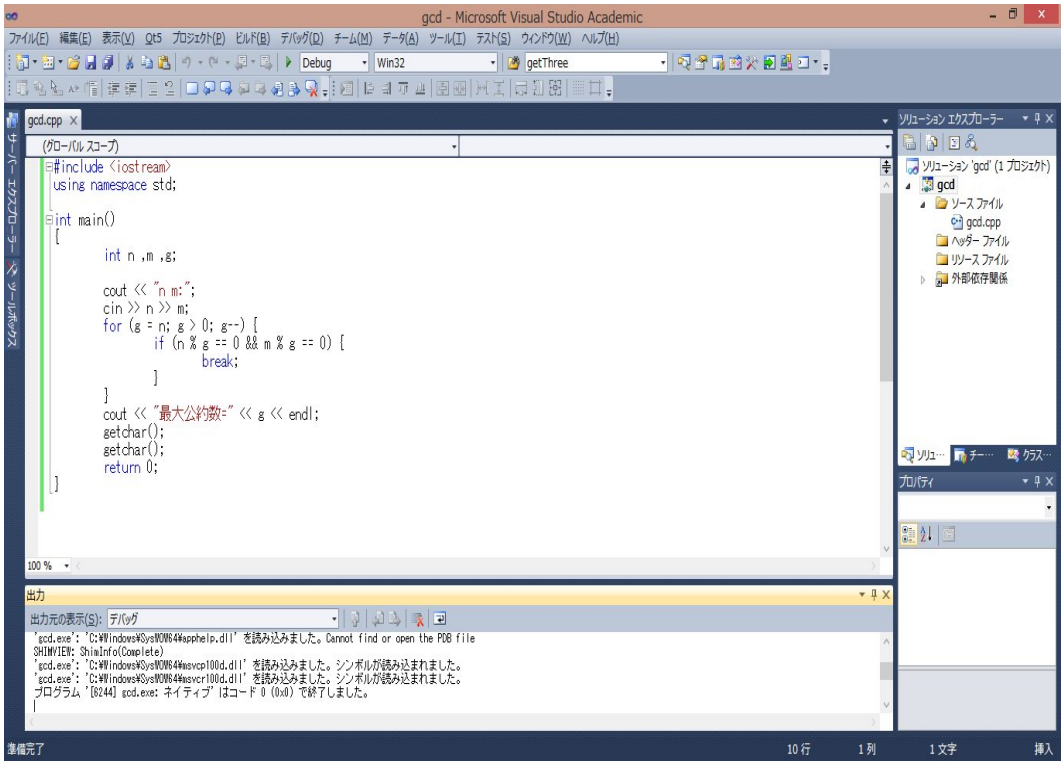
となります。最後に、cin を使ったので、

```
        getchar();
```

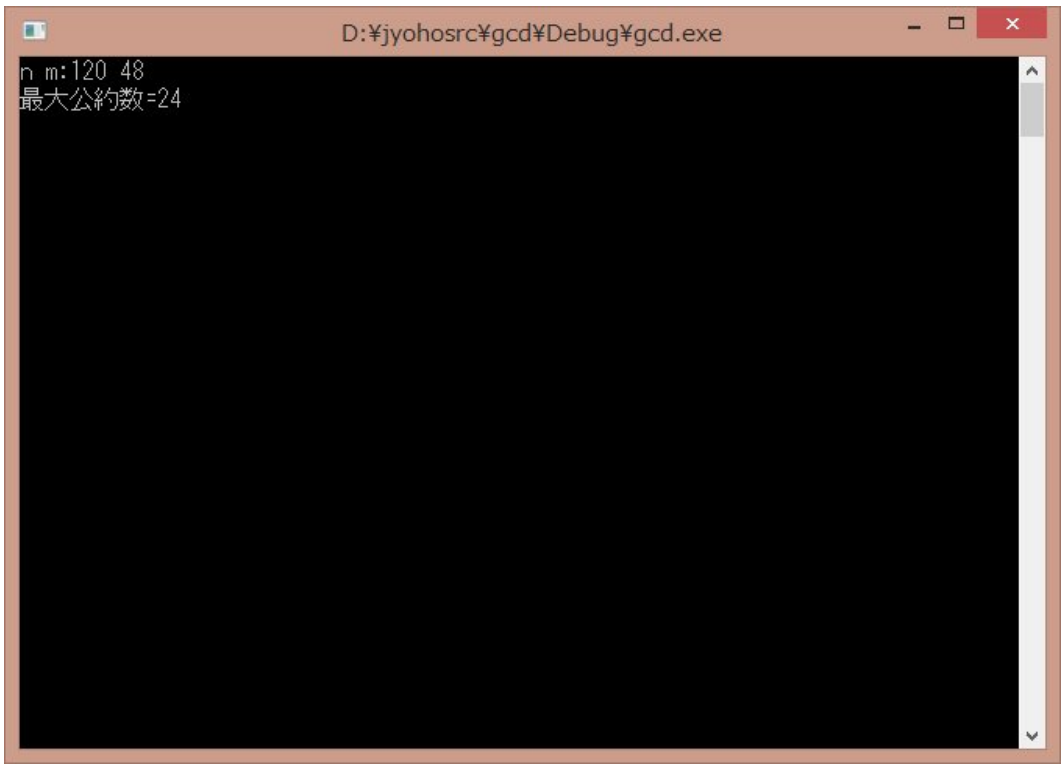
を追加します。

```
#include <iostream>
using namespace std;

int main()
{
        int n, m, g;

        cout << "n m:";
        cin >> n >> m;
        for (g = n; g > 0; g--) {
                if (n % g == 0 && m % g == 0) {
                        break;
                }
        }
        cout << "最大公約数=" << g << endl;
        getchar();
        getchar();
        return 0;
}
```

となります。プログラムが完成しました。



実行してみます。



これは次のようにプログラミングすることもできます。

```
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
    int n, m;

    cout << "n m:";
    cin >> n >> m;
   while (m != 0) {
        int temp = n;
        n = m;
        m = temp % m;
    }
    cout << "最大公約数=" << n << endl;
    getchar();
    getchar();
    return 0;
}
```

while 文を使っています。このアルゴリズムを「ユークリッドの互除法」といいます。
while 文は

```
while (条件式) {
      命令の並び
}
```

の形式で、条件式が真 (true) の間は{ と } の間の命令達を繰り返します。今の場合、m が 0 でない間は { と } の間の命令達を繰り返します。

```
   while (m != 0) {
        int temp = n;
        n = m;
        m = temp % m;
    }
```

の

```
        int temp = n;
        n = m;
        m = temp % m;
```

で、

```
        int temp = n;
```

とありますが、int の変数 temp を宣言して、 n を代入しています。C++ では変数を使う直前に変数を宣言することも可能です、この場合、変数 temp は while 文の { と } の中だけで使用することが可能です。ここでしていることは、m の値を n に代入し、m には n を m で割った余りを入れますが、

```
        n = m;
        m = n % m;
```

とそのまますると n の値が m になって、n を m で割った余りは常に 0 になってしまうので、一時変数 temp に n の値を保存してから、 n に m を代入し、temp を割った余りを m に代入しています。私たちはこのアルゴリズムを小学生にときに習いました。

Prolog や Scheme のように再帰的関数を使って、

```
#include <iostream>
using namespace std;

int gcd(int n, int m) {
    if (m == 0) {
        return n;
    } else {
        return gcd(m, n % m);
    }
}

int main(int argc, char* argv[])
{
    int n, m, g;

    cout << "n m: ";
    cin >> n >> m;
    cout << "最大公約数=" << gcd(n, m) << endl;
    getchar();
    getchar();
    return 0;
}
```

としても良いです。

```
int gcd(int n, int m) {
    if (m == 0) {
        return n;
    } else {
        return gcd(m, n % m);
    }
}
```

で、gcd() という int の値を返す int の値を代入して呼ばれる引数 n と int の値を代入して呼ばれる引数 m の二つの引数を持つ関数が定義されています。関数も使われる前に宣言されている必要があります。Pythin や Sheme などと比べるとどのような値を返すか、引数にはどんな値が与えられるか全部明確にして置かなければいけないので面倒ですが、プログラムの実行速度が違います。高速性が要求される複雑なプログラムは C や C++ で作ります。私のように趣味でプログラム作るには Python や Sheme が良いです。仕事でプログラムを作るのは、プログラムは作った後はコストがかからないので、今はグローバルな競争で、給料の安いインドや中国やベトナムの人たちとの競争になるので大変です。他人の真似の出来ない技術を必死になって身に着けないとやっていけません。IT 企業がブラック企業だといわれる所以です。

次の例は素因数分解の BASIC のプログラムです。これも旺文社の昔の数学 A の教科書で見つけたプログラムです。

```
10 REM 素因数分解
20 INPUT N
30 M=1
40 M=M+1
50 IF INT(N/M)*M<N THEN 40
60 PRINT M;
70 N=N/M
80 IF N>1 THEN 50
90 END
```

これも次のようにプログラミングするのが良いです。

```
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
    int n, k;

    cout << "n=";
    cin >> n;
    k = 2;
    while (n > 1) {
        while (n % k == 0) {
            cout << k << " ";
            n = n / k;
        }
        k++;
    }
    cout << endl;
    getchar();
    getchar();
```

```
    return 0;
}
```

while 文が入れ子になっています。
ここで使われているアルゴリズムは

1. N を入力する
2. K を 2 とする
3. N が 1 より大きい間は次を繰り返す
   (a) N が K で割り切れる間は K を表示し、N を K で割る
   (b) K をひとつ増やす

です。
プログラムを作るときは、何をすべきか箇条書きにし、それをだんだんプログラムコードに近づけていきます。まず何をすべきか言葉で表現し、それを実現するには何をすればいいか箇条書きにし、そのそれぞれのステップを実現するには何をすればいいかそれぞれ箇条書きにし、を繰り返し、プログラムに近づけます。これをトップダウンの方法と言います。他人の書いたプログラムを読むときは各ステップが何をやっているか解析し、まとまりのあるいくつかのステップが何をやっているか解析し、段々大きく括って何をしているか調べ、最終的に何をしているか調べます。これをボトムアップの方法と言います。
例題：ヘロンの公式を使って、三角形の面積を計算するプログラムを作れ。

```
#include <iostream>
#include <math.h>

using namespace std;

int main(int argc, char* argv[])
{
    double a, b, c;

    cout << "a="; cin >> a;
    cout << "b="; cin >> b;
    cout << "c="; cin >> c;
    if (a+b<=c || b+c<=a || c+a<=b) {
        cout << "三角形ではありません" << endl;
        getchar();
        return 1;
    }
    double S = (a+b+c)/2;
    S = sqrt(S*(S-a)*(S-b)*(S-c));
    cout << "面積は" << S << "です" << endl;
    getchar();
```

```
    getchar();
    return 0;
}
```

```
    if (a+b<=c || b+c<=a || c+a<=b) {
        cout << "三角形ではありません" << endl;
        getchar();
        return 1;
    }
```

の if 文は、$a+b$ が　c　以下（a+b<=c) か $b+c$ が　a　以下（b+c<=a) か $c+a$ が　b　以下（c+a<=b) の時という意味です。「または」を表す論理演算子は || です。以下と以上を表す比較演算子は<= と >= です。否定を表す論理演算子は ! です。

```
    if (a+b<=c || b+c<=a || c+a<=b) {
    }
```

は

```
    if (!(a+b > c && b+c > a &&  c+a> b)) {
    }
```

と表現することも出来ます。

100000 以下の友愛数をすべて求めてみよう。友愛数（ゆうあいすう）とは、異なる 2 つの自然数の組で、自分自身を除いた約数の和が、互いに他方と等しくなるような数をいう。親和数とも呼ばれる。

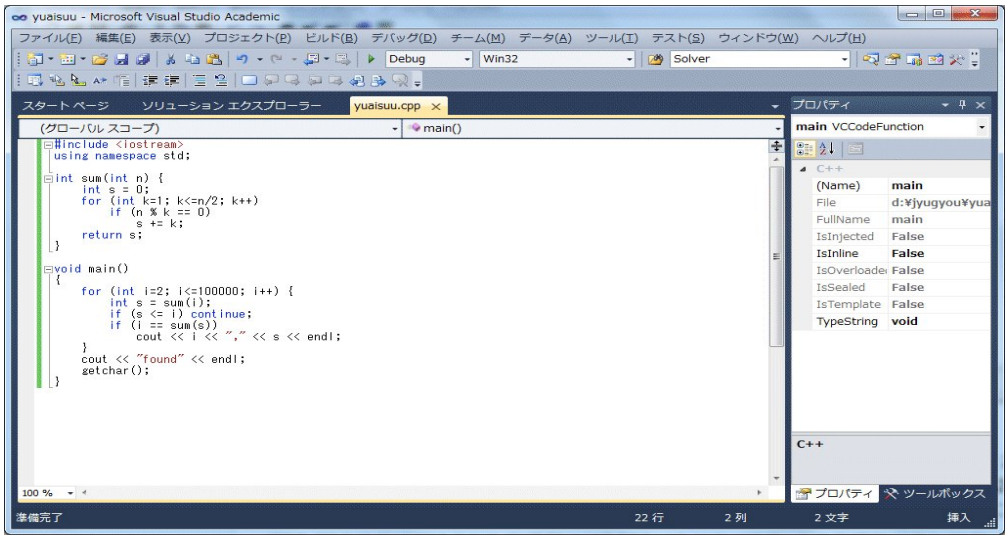
一番小さな友愛数の組は (220, 284) である。220 の自分自身を除いた約数は、1,2,4,5,10,11,20,22,44,55,110 で、和は 284 となる。一方、284 の自分自身を除いた約数は、1,2,4,71,142 で、和は 220 である。

上と同様にして、yuaisuu.cpp を作り、

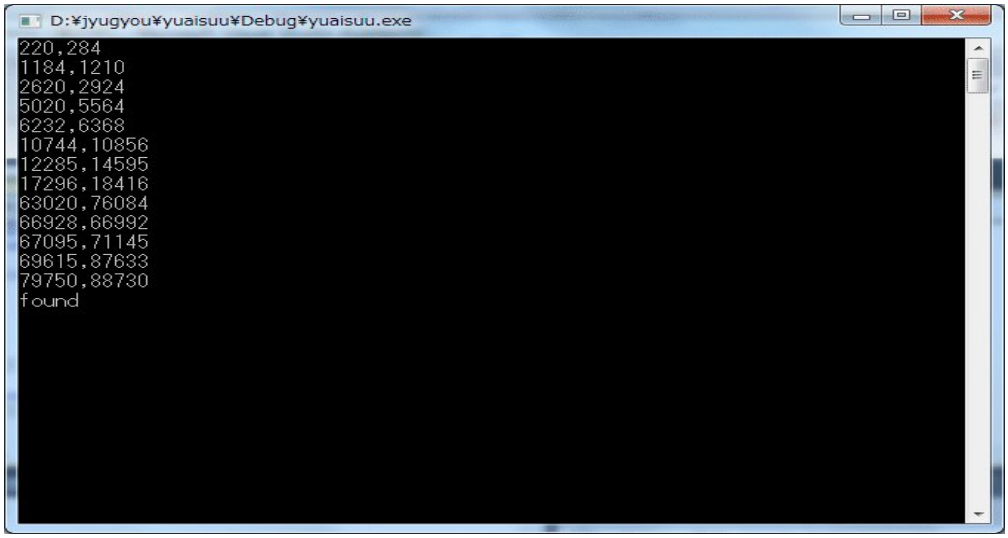
```
#include <iostream>
using namespace std;

int sum(int n) {
    int s = 0;
    for (int k=1; k<=n/2; k++)
        if (n % k == 0)
            s += k;
    return s;
}

void main()
{
    for (int i=2; i<=100000; i++) {
        int s = sum(i);
        if (s <= i) continue;
```

```
        if (i == sum(s))
            cout << i << "," << s << endl;
    }
    cout << "found" << endl;
    getchar();
}
```

と打ち込む。



実行する。



プログラムは

```
#include <iostream>
using namespace std;
```

は、おきまりのもので、いつでもこうします。

```
int sum(int n) {
    int s = 0;
    for (int k=1; k<=n/2; k++)
        if (n % k == 0)
            s += k;
    return s;
}
```

が、整数 n の自分自身以外の約数の和を計算する関数です。関数 sum() は、int の型の値を返す関数で、関数名は sum で、int 型の引数を 1 つ取ります。

```
    int s = 0;
    for (int k=1; k<=n/2; k++)
        if (n % k == 0)
            s += k;
    return s;
```

が、関数の本体で、まず、int 型の変数（整数値を保持する変数）s を宣言し、初期値を 0 にしています。このようなものを変数の定義と言います。これを

```
    int s ;
    s = 0;
```

と、s の宣言と s への代入の 2 行にしても良いです。変数 s を使う場合、必ず使う前に、宣言もしくは定義しておく必要があります。以下、足し算をしていくので、加法のゼロ元である 0 に s を初期化しています。次に、for 文で、繰り返しを表現しています。

```
    for (int k=1; k<=n/2; k++)
        if (n % k == 0)
            s += k;
```

で、整数変数 k を 1 から (k=1)、 n/2 まで（k<=n/2）、k を 1 つづつ増やしながら (k++)、n が k で割り切れるなら（if (n % k == 0)）、s に k を加えて（s += k;）います。この for 文が終わると s に n の自分以外のすべての約数の和がセットされます。最後に

```
    return s;
```

と return 文で s を関数の値として返しています。

```
void main()
{
    for (int i=2; i<=100000; i++) {
        int s = sum(i);
        if (s <= i) continue;
        if (i == sum(s))
            cout << i << "," << s << endl;
    }
    cout << "found" << endl;
```

```
    getchar();
}
```

が main() 関数です。整数 i を 2 から 100000 まで一つづつ増加させながら、i の自分自身以外の約数の和を計算し、 s に代入し、s が i 以下なら continue 文で以下の文を飛ばして次の i を調べることとし、s が i より大きければ、s の自分自身以外の約数の和を計算して、i と比較します。もし、i と s の自分自身以外の約数の和が等しければ、i と s が友愛数なので、i と s を表示します。計算に時間がかかるので、for 文が修了すれば, found と表示させています。ちなみに、i == sum(i) となる i を完全数といいます。

問題： 100000 以下の完全数を計算するプログラムを作れ。

例題：Zeller の公式を使って、西暦の年月日を入力して曜日を計算するプログラムを作れ。

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

using namespace std;

int main(int argc, char* argv[])
{
    int y, m, d, w;

    cout << "西暦年:"; cin >> y;
    cout << "月:"; cin >> m;
    cout << "日:"; cin >> d;
    getchar();
    m = m - 2;
    if (m < 1) {
        y--; m += 12;
    }
    int b = y % 100;
    int c = y / 100;
    w = ((26*m-2)/10+d+b+b/4+5*c+c/4) % 7;
    switch(w) {
        case 0: cout << "日曜日です" << endl; break;
        case 1: cout << "月曜日です" << endl; break;
        case 2: cout << "火曜日です" << endl; break;
        case 3: cout << "水曜日です" << endl; break;
        case 4: cout << "木曜日です" << endl; break;
        case 5: cout << "金曜日です" << endl; break;
        case 6: cout << "土曜日です" << endl; break;
    }
    getchar();
    return 0;
}
```

```
switch(w) {
    case 0: cout << "日曜日です" << endl; break;
    case 1: cout << "月曜日です" << endl; break;
    case 2: cout << "火曜日です" << endl; break;
    case 3: cout << "水曜日です" << endl; break;
    case 4: cout << "木曜日です" << endl; break;
    case 5: cout << "金曜日です" << endl; break;
    case 6: cout << "土曜日です" << endl; break;
}
```

は、w が 0 なら日曜日ですと表示し、w が 1 なら月曜日ですという様に w の値で場合分けします。

演習問題

1. ある都市の中型タクシーの料金は、2 km までは 500 円、あとは 300 m までごとに 80 円増すことになっている。乗車距離 x (km) を入力し、料金 y (円) を表示するプログラムを作れ。
2. 西暦年を入力して閏年かどうかを判定するプログラムを作れ。１５８３年以降グレゴリオ暦が使われていて、閏年は、４で割り切れるが１００の倍数で無いときである。ただし４００で割り切れれば閏年である。
3. a,b,c が定数、$a \neq 0$ のとき a,b,c を入力して、二次方程式 $ax^2 + bx + c = 0$ の解の個数を求めるプログラムを作れ。
4. 三角形 ABC の 3 辺の長さ a, b, c を与えこれから三角形 ABC が鋭角三角形、直角三角形、鈍角三角形のいずれであるかを判定するプログラムを作れ。

例題：整数を入力し、素数かどうかを判定するプログラムを作れ。

これは色々な作り方がありますが、単純なアルゴリズムは次のようなものです。

1. N を入力する
2. I を 2, 3, 4, 5, $\cdots$, と $\sqrt{N}$ まで変化させ、I が N を割り切るなら N は合成数
3. 上のチェックで合成数でなければ素数

これをもっと具体的にします。

N を入力するは

```
int n;
cout << "n="; cin >> n;
```

で表現できます。文は一行に複数書いても良いです。

I を 2, 3, 4, 5, $\cdots$, と $\sqrt{N}$ まで変化させ、

は for 文で表現できます。

```
int i;
for (i=2; i <= sqrt((double)n); i++) {
}
```

とすれば良いです。sqrt() は引数に double の値を取り、 n は int の値を保持する変数ですから、n の値を double の値に変換してもらうよう (double)n とキャストしています。
　I が N を割り切るならは if 文を使います。

```
        if ( n % i == 0) {
        }
```

です。
　N は合成数の部分は

```
        if ( n % i == 0) {
            cout << n << "は" << i << "で割り切れます" << endl;
            return 0;
        }
```

です。
　上のチェックで合成数でなければ素数の部分は

```
      cout << n << "は素数です" << endl;
```

です。これらをまとめると

```
#include <iostream>
#include <math.h>

using namespace std;

int main()
{
    int n;
    cout << "n="; cin >> n;
    int i;
    for (i=2; i <= sqrt((double)n); i++) {
        if ( n % i == 0) {
            cout << n << "は" << i << "で割り切れます" << endl;
            return 0;
        }
    }
     cout << n << "は素数です" << endl;
    return 0;
}
```

コマンドプロンプトで実行するプログラムはこれで完成です。
　最後にキーを入力するまで画面が閉じないようにするには

```
#include <iostream>
#include <math.h>

using namespace std;

int main()
{
    int n;
    cout << "n="; cin >> n;
    int i;
    for (i=2; i <= sqrt((double)n); i++) {
        if ( n % i == 0) {
            cout << n << "は" << i << "で割り切れます" << endl;
            getchar();
            getchar();
            return 0;
        }
    }
     cout << n << "は素数です" << endl;
    getchar();
    getchar();
    return 0;
}
```

これで完成です。
　これは次のように関数を使ってもいいです。

```
#include <iostream>
#include <math.h>

using namespace std;

bool primeP(int n)
{
    for (int i=2; i<=sqrt((double)n); i++) {
        if (n % i == 0)
            return false;
    }
    return true;
}

int main(int argc, char* argv[])
{
    int n;
```

```
    cout << "n="; cin >> n;
    if (primeP(n))
        cout << "素数です" << endl;
    else
        cout << "合成数です" << endl;
    getchar();
    getchar();
    return 0;
}
```

bool は true か false の値をとる真理値の型です。
関数は

```
bool primeP(int n)
{
    for (int i=2; i<=sqrt((double)n); i++) {
        if (n % i == 0)
            return false;
    }
    return true;
}
```

のように、

```
戻り値の型　関数名 (引数の並び)
{
    関数の本体
}
```

からなります。上の関数は整数値である 1 つの引数 n を取り、真理値 true か false　を戻す関数です。関数は使う前に宣言しておく必要があります。今の場合、primeP() が使われるのは main() 関数で、 main() 関数の前に primeP() が定義されているので、これで良いですが、もし

```
#include <iostream>
#include <math.h>

using namespace std;

bool primeP(int n);

int main(int argc, char* argv[])
{
    int n;

    cout << "n="; cin >> n; getchar();
```

```
    if (primeP(n))
        cout << "素数です" << endl;
    else
        cout << "合成数です" << endl;
    getchar();
    return 0;
}

bool primeP(int n)
{
    for (int i=2; i<=sqrt((double)n); i++) {
        if (n % i == 0)
            return false;
    }
    return true;
}
```

のように、 main() 関数の後ろに primeP() が定義されている場合は、main() 関数の前に

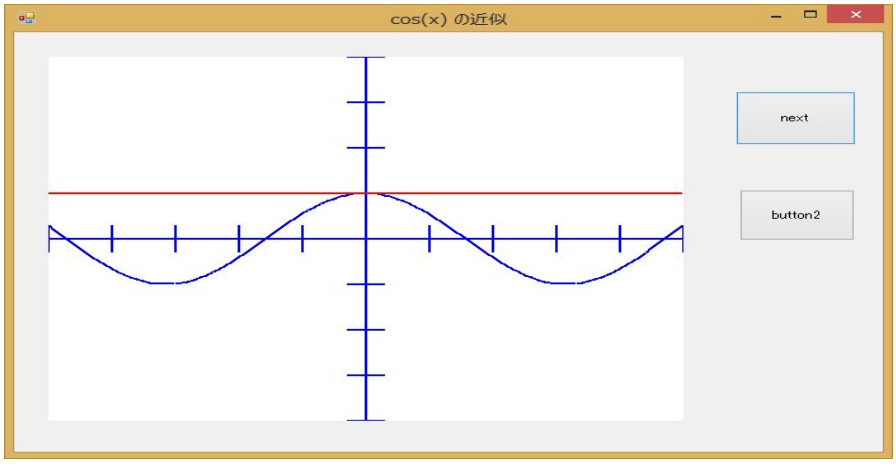
```
bool primeP(int n);
```

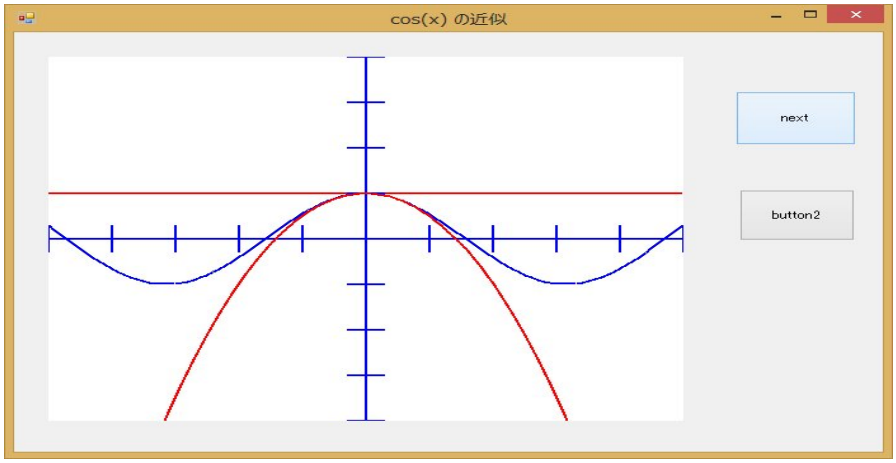
のように primeP() がどのような引数を取り、どのような戻り値を返すか宣言しておく必要があります。bool primeP(int n) 関数は、整数値 n が i=2 から $\sqrt{n}$ まで i を 1 つづつ増やしながら、どれかの i が n を割り切るなら false を返し、すべて割り切らなければ true を返す関数として定義しています。

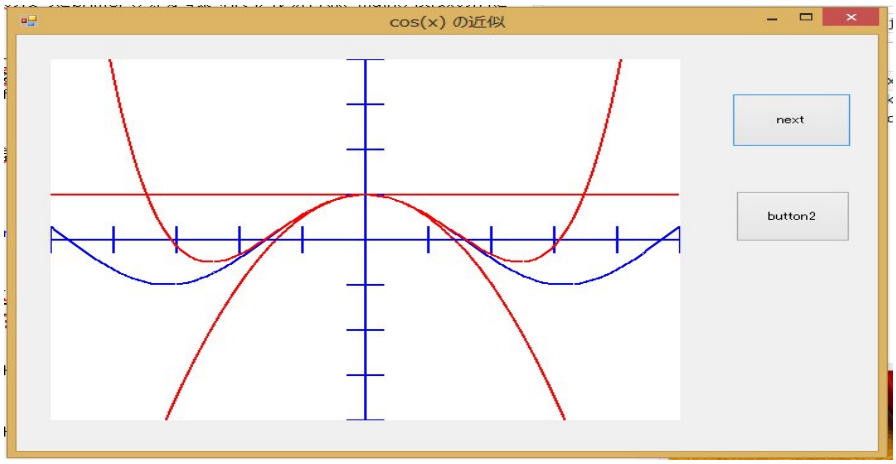
例題： 解析学で学ぶように、$\cos(x)$ や $\sin(x)$ は

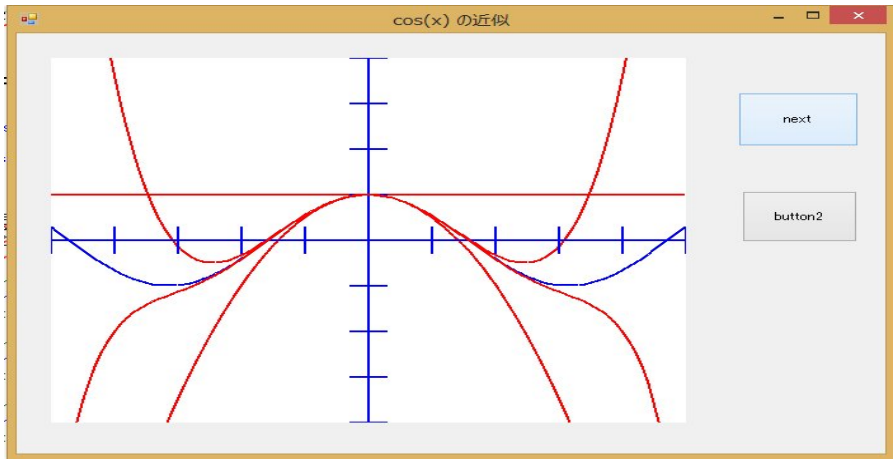
$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}, \qquad \sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

と級数を使って表すことができます。x はラジアンです。この級数はすべての x に対して定義されていて非常に計算効率が良いです。これを使って、$\cos(x)$ を計算するプログラムを作れ。









のように

$$cosN(N,x) = \sum_{n=0}^{N} \frac{(-1)^n}{(2n)!} x^{2n}$$

は　$\cos(x)$ に近づいていきます。$\cos(x)$ を

$$cosN(N,x) = \sum_{n=0}^{N} \frac{(-1)^n}{(2n)!} x^{2n}$$

で近似するわけですが、x により N の値をなんにすればよいかが決まります。
それで、 N の値を決めるには、例えば、

$$|cosN(N,x) - cosN(N-1,x)| < 0.0000001$$

となる $cosN(N,x)$ を答えとすれば良いです。
これをそのままプログラムにすると

```
#include <iostream>
#include <math.h>

using namespace std;

double fact(double n) {
    double r = 1;
    for (int i=1; i<=n; i++)
        r *= i;
    return r;
}

double cosN(int n, double x) {
    double s = 0;
    for (int i=0; i<=n; i++) {
        if (i % 2 == 0)
            s += pow(x, 2*i) / fact(2*i);
        else
            s -= pow(x, 2*i) / fact(2*i);
    }
    return s;
}

int main()
{
    double x, y;
    int N = 0;

    cout << " X= ";
    cin >> x;
    y = x * 4 * atan(1.0) / 180;

    do {
        N++;
    } while (fabs(cosN(N, y) - cosN(N-1, y)) > 0.0000001);
    cout <<  N << "回目 : cos(" << x << ")= " << cosN(N, y) << endl;
```

```
    getchar(); getchar();
    return 0;
}
```

となります。

```
#include <iostream>
#include <math.h>

using namespace std;
```

は、出力 cout とべき乗を与える数学関数 pow() と逆正接関数 atan() を使うために必要な処置です。

```
double fact(double n) {
    double r = 1;
    for (int i=1; i<=n; i++)
        r *= i;
    return r;
}
```

は、n の階乗を計算する関数の定義です。階乗を計算する関数 double fact(double n)

```
double fact(double n) {
    double r = 1;
    for (int i=1; i<= n; i++)
        r *= i;
    return r;
}
```

は 4 つの部分から構成されています。関数は、戻り値の型　関数名 (引数の並び) {関数の本体} の構成になっています。今の場合は、戻り値の型は double 、すなわち実数値を返す関数であると宣言しています。関数名は fact です。引数の並びは今の場合、引数は一個で、double n と実数値を取る引数 n があることを示しています。関数の本体は { と } で囲まれた部分で、今の場合、

```
    double r = 1;
    for (int i=1; i<= n; i++)
        r *= i;
    return r;
```

が関数の本体で、実数値を取る変数 r の定義 (r は実数値変数であるという宣言とその値を 1 とするという初期設定が同時に行われています) があって、次の for 文で、整数値変数 i を 1 から n まで 1 つづつ増やしながら、r に順次掛けています。この結果、for 文が修了したとき、r は n の階乗の値になっています。最後に、r の値を関数の値として return 文で返しています。

何度も書きますが、for 文は

```
for (初期設定; 条件式; 後処理) {
    命令の並び
}
```

の形式で、まず、初期設定を実行し、条件式が真であれば、命令の並びを実行し、後処理をし、条件式が真であれば、命令の並びを実行し、後処理をし、を条件式が偽になるまで繰り返します。条件式の一回目の判定が偽であれば、命令の並びは一回も実行されません。

```
for (初期設定; 条件式; 後処理) {
    命令の並び
}
```

で、命令の並びが 1 つの命令の場合は

```
    for (int i=1; i<= n; i++)
        r *= i;
```

や

```
    for (int i=0; i<=n; i++)
        if (i % 2 == 0)
            s += pow(x, 2*i) / fact(2*i);
        else
            s -= pow(x, 2*i) / fact(2*i);
```

のように、命令の並びを囲む { と } は省略できます。

```
double cosN(int n, double x) {
    double s = 0;
    for (int i=0; i<=n; i++) {
        if (i % 2 == 0)
            s += pow(x, 2*i) / fact(2*i);
        else
            s -= pow(x, 2*i) / fact(2*i);
    }
    return s;
}
```

は、cos(x) の級数展開の第 n 項までの和を計算する関数の定義です。

```
        if (i % 2 == 0)
            s += pow(x, 2*i) / fact(2*i);
        else
            s -= pow(x, 2*i) / fact(2*i);
```

は、if 文で、i を 2 で割った余りが 0 ならば、s に pow(x, 2*i) / fact(2*i) を加え、そうでなければ、s から pow(x, 2*i) / fact(2*i) を引くことを指示しています。

```
        y = x * 4 * atan(1.0) / 180;
```

は、入力した x をラジアン y に変換しています。 4 * atan(1.0) で円周率を与えます。atan(double x) はアークタンジェントを与える組み込み関数です。

```
    do {
        N++;
    } while (fabs(cosN(N, y) - cosN(N-1, y)) > 0.0000001);
```

は、 do while 文で、

```
       do {
            命令の並び
} while (条件式);
```

の形式で、while(条件式) で与えられる条件式が真 (true) である間は { と } の間の命令達を繰り返します。この場合、少なくとも一回は { と } の間の命令達を実行します。fabs() は絶対値を計算する組み込み関数（VC++ で前もって定義されている数学関数）です。do while 文の前で、

```
    int N = 0;
```

と、N の宣言とN の値を 0 に初期設定しています。そして、

```
        N++;
```

で、N を一つ増やし 、

```
    } while (fabs(cosN(N, y) - cosN(N-1, y)) > 0.0000001);
```

で、 $cosN(N, y)$ と $cosN(N-1, y)$ の差の絶対値が 0.0000001 より大きいかどうかを調べています。$cosN(N, y)$ と $cosN(N-1, y)$ の差の絶対値が 0.0000001 より大きければ、元に戻って、N を一つ増やし 、 $cosN(N, y)$ と $cosN(N-1, y)$ の差の絶対値が 0.0000001 より大きいかどうかを調べます。これを $cosN(N, y)$ と $cosN(N-1, y)$ の差の絶対値が 0.0000001 以下になるまで繰り返します。

最後に

```
    cout <<  N << "回目 : cos(" << x << ")= " << cosN(N, y) << endl;
```

で、結果を表示しています。

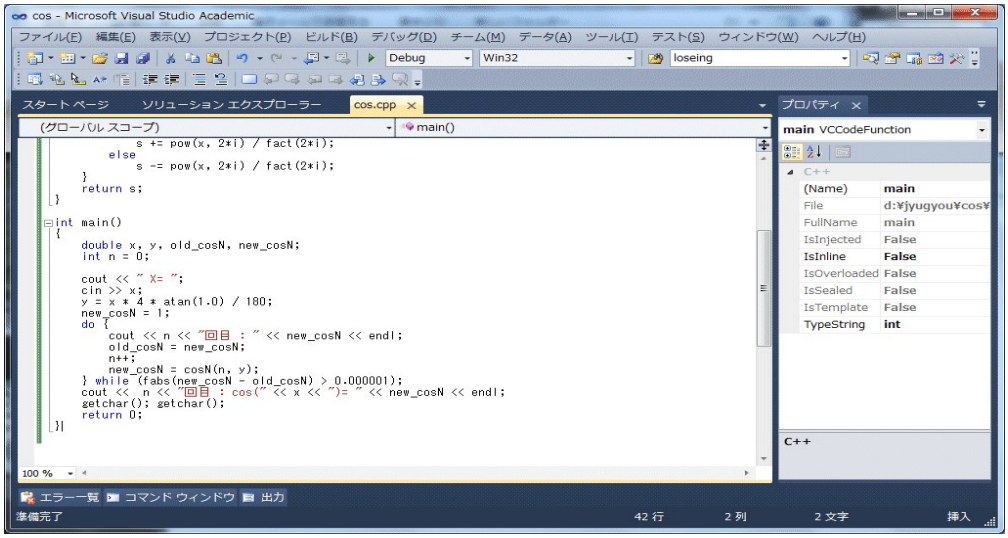
これで問題は解決した訳ですが、実は問題があります。それぞれ $cosN(N, y)$ を二度づつ計算しています。$cosN(N, y)$ を計算するのに時間がかかる場合は、このような無駄は避けるように普通はします。

このためには、例えば、

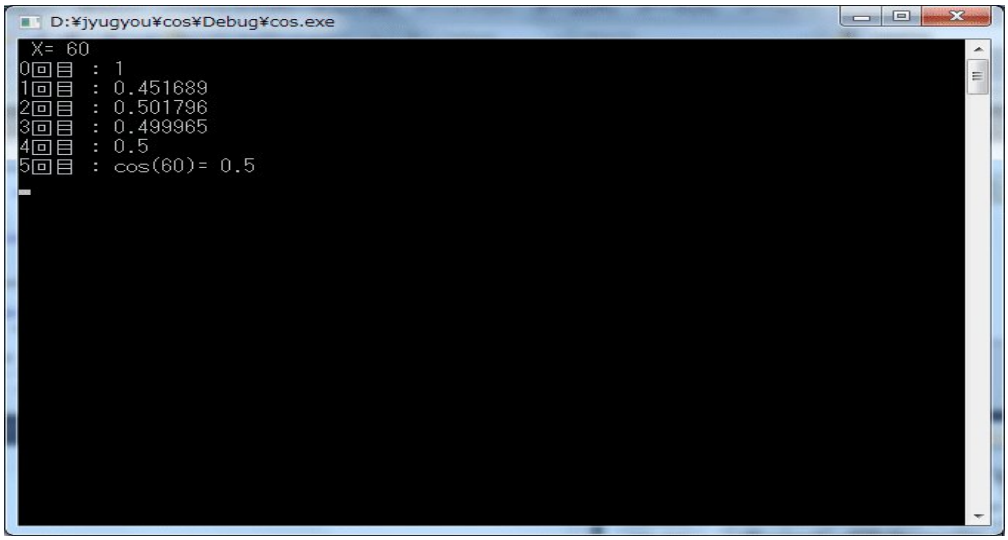
```
#include <iostream>
#include <math.h>

using namespace std;

double fact(double n) {
    double r = 1;
```

```
    for (int i=1; i<=n; i++)
        r *= i;
    return r;
}

double cosN(int n, double x) {
    double s = 0;
    for (int i=0; i<=n; i++) {
        if (i % 2 == 0)
            s += pow(x, 2*i) / fact(2*i);
        else
            s -= pow(x, 2*i) / fact(2*i);
    }
    return s;
}

int main()
{
    double x, y, old_cosN, new_cosN;
    int n = 0;

    cout << " X= ";
    cin >> x;
    y = x * 4 * atan(1.0) / 180;
    new_cosN = 1;
    do {
        cout << n << "回目 : " << new_cosN << endl;
        old_cosN = new_cosN;
        n++;
        new_cosN = cosN(n, y);
    } while (fabs(new_cosN - old_cosN) > 0.000001);
    cout <<  n << "回目 : cos(" << x << ")= " << new_cosN << endl;
    getchar(); getchar();
    return 0;
}
```

とプログラミングできます。



実行し、60 を入力すると



となる。Enter キーを押すとウィンドが閉じる。

do while 文の前で、

```
new_cosN = 1;
```

と、級数の初項を new_cosN にセットしています。そして、

```
cout << n << "回目 : " << new_cosN << endl;
```

で、new_cosN を表示し、

```
old_cosN = new_cosN;
```

で、現在の new_cosN の値を old_cosN に保存し、

```
n++;
```

で、n を一つ増やし、

```
        new_cosN = cosN(n, y);
```

で、次の項までの級数の値を new_cosN にセットしています。

```
    } while (fabs(new_cosN - old_cosN) > 0.000001);
```

で、一つ前の値 old_cosN と新しい値 new_cosN の差の絶対値が 0.000001 より大きい間はこの操作を繰り返しています。

更に、

```
double cosN(int n, double x) {
    double s = 0;
double t = 1;
    for (int i=0; i<=n; i++) {
        if (i % 2 == 0)
            s += t;
        else
            s -= t;
        t = t * x * x / (2*i+1) / (2*i+2);
    }
    return s;
}
```

と定義すれば、

```
double fact(double n) {
    double r = 1;
    for (int i=1; i<=n; i++)
        r *= i;
    return r;
}
```

の定義が不要になり、効率が良くなります。

```
#include <iostream>
#include <iomanip>
#include <math.h>

using namespace std;

inline ios_base& general(ios_base& b)
{
    b.setf(ios_base::fmtflags(0), ios_base::floatfield);
    return b;
}
```

```
double fact(double n) {
    double r = 1;
    for (int i=1; i<=n; i++)
        r *= i;
    return r;
}

double cosN(int n, double x) {
    double s = 0;
    for (int i=0; i<=n; i++) {
        if (i % 2 == 0)
            s += pow(x, 2*i) / fact(2*i);
        else
            s -= pow(x, 2*i) / fact(2*i);
    }
    return s;
}

int main()
{
    double x, y, old_cosN, new_cosN;
    int n = 0;

    cout << " X= ";
    cin >> x;
    y = x * 4 * atan(1.0) / 180;
    new_cosN = 1;
    do {
        cout << n << "回目 : " << fixed << setprecision(12) << new_cosN << endl;
        old_cosN = new_cosN;
        n++;
        new_cosN = cosN(n, y);
    } while (fabs(new_cosN - old_cosN) > 0.000001);
    cout << general << n << "回目 : cos(" << x << ")= " << fixed << setprecision(12)
           <<new_cosN << endl;
    getchar(); getchar();
    return 0;
}
```

と修正すると表示の精度を変えることが出来ます。何処が変わったかというと

```
#include <iomanip>
```

の追加（setprecision() を使うためにこれが必要です）と

```
inline ios_base& general(ios_base& b)
{
    b.setf(ios_base::fmtflags(0), ios_base::floatfield);
    return b;
}
```

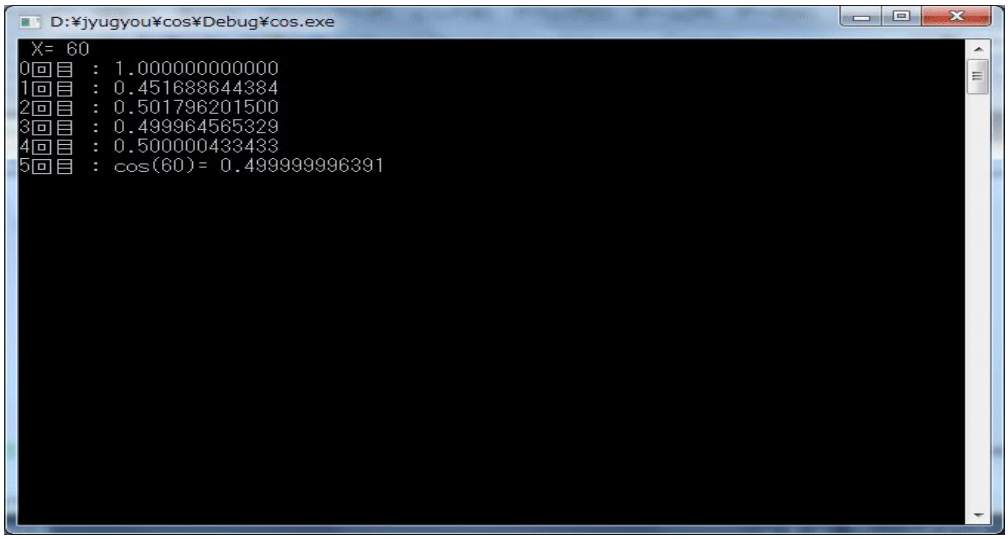
の追加（fixed や setprecision() を廃止し、デフォルトの状態に戻すために必要です。cos(60) の 60 を 60.0000000000 ではなく、60 と表示するためにこうしています。これは何も考えず、このようにするものだとしておいて下さい。あらゆる事をいっぺんに理解しようとしてはいけません。魔法を使うための呪文です。）と

```
        cout << n << "回目 : " << fixed << setprecision(12) << new_cosN << endl;
```

と

```
    cout << general << n << "回目 : cos(" << x << ")= " << fixed << setprecision(12)
          <<new_cosN << endl;
```

の修正をしています。setprecision(12) で 12 桁の精度で、また fixed でそのまま（固定小数点を使用する）表示するようにします。実行すると



のような表示を得ます。fixed では、1234.56789 は 1234.567890 のように固定小数点を使用して表示されますが、scientific を使うと 1.234568e+003 のように仮数表記と指数表記を使用して表示されます。general フォーマット (デフォルト）は general の精度内で、数値的に最も正確な表現が得られる fixed または scientific が選択されます。

演習問題

1. 整数を入力しその約数をすべて表示するプログラムを作れ。
2. $\cos(x)$ を計算するプログラムを真似て、 $\sin(x)$ を計算するプログラムを作れ。
3. ライプニッツの級数を利用して、円周率を計算するプログラムを作れ。

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

がライプニッツの級数です。本当はグレゴリーの級数という方が正しいそうで、歴史的には余りにも有名ですが、円周率の計算には向かない。

4. 自然数 n について、入力された n の各桁の数字の和を求めるプログラムを作れ。
5. $a^2+b^2=c^2$ となる自然数 a, b, c の組みとその個数を求めるプログラムを作れ。但し $c<100$ とする。
6. 入力された自然数 a, b の公約数の総和を求めるプログラムを作れ。

0.2 配列

いままで扱った変数には一つの数値しか記憶できない。多数の数値をまとめて効率よく扱うには Python や Prolog や Scheme ではリストを使いましたが、 C++ では配列を使います。配列の宣言は変数名の後ろに [と] で配列の大きさを囲む。

```
    int a[10];
```

は整数型のデータ記憶場所を 10 個準備し、a[0],a[1],a[2], ... ,a[9] でアクセスできる。a[0] から a[9] であることに注意。次はフィボナッチ数列の値を求める配列を用いたプログラムである。

```
#include <iostream>

using namespace std;

int main()
{
    int f[21], i;

    f[0] = 1;
    f[1] = 1;
    for (i=2; i <= 20; i++)
        f[i] = f[i-1] + f[i-2];
    for (i=0; i <= 20; i++)
        cout << i << " " <<  f[i] << endl;
    getchar();
    return 0;
}
```

```
    int f[21], i;
```

は int の配列 f と変数 i を宣言している。int f[21]; なので f[0], f[1], f[2], ⋯, f[20] の 2 1 個の int の値を入れる場所が確保される。

```
    f[0] = 1;
    f[1] = 1;
```

で、 f[0] と f[1] に 1 を代入している。

```
    for (i=2; i <= 20; i++)
        f[i] = f[i-1] + f[i-2];
```

で、f[2] = f[1]+f[0], f[3] = f[2]+f[1], f[4] = f[3]+f[2], $\cdots$, f[20] = f[19]+f[18] と順番に配列に値を代入している。f[2] = f[1]+f[0] は f[1] と f[0] に保存されている値の和を f[2] に代入することを表している。

```
    for (i=0; i <= 20; i++)
        cout << i << " " <<  f[i] << endl;
```

で、i = 0, 1, 2, 3, $cdots$, 20 に対して、i と f[i] を表示している。
　次はベクトルの外積を計算するプログラムである。

```
#include <iostream>

using namespace std;

void vp(double a[], double b[], double c[])
{
    c[0] = a[1] * b[2] - a[2] * b[1];
    c[1] = a[2] * b[0] - a[0] * b[2];
    c[2] = a[0] * b[1] - a[1] * b[0];
}
int main()
{
    double a[3], b[3], c[3];

    cout << "ベクトル 1 の x, y, z?";
    cin >> a[0] >> a[1] >>a[2];
    cout << "ベクトル 2 の x, y, z?";
    cin >>b[0] >> b[1] >>b[2];
    vp(a, b, c);
    cout << "外積=" << c[0] << " " << c[1] << " " << c[2] << endl;
    getchar();
    getchar();
    return 0;
}
```

配列は

```
void vp(double a[], double b[], double c[])
{
    c[0] = a[1] * b[2] - a[2] * b[1];
    c[1] = a[2] * b[0] - a[0] * b[2];
    c[2] = a[0] * b[1] - a[1] * b[0];
}
```

のように関数に渡すこともできる。このときアドレス渡しとなる。すなわち、関数内で値を変更すると実引数も値がその値になる。もう少し説明すると

```
#include <iostream>

using namespace std;

void func(int a)
{
    a = 6;
}

int main()
{
    int n = 10;
    func(n);
    cout << n << endl;
    getchar();
}
```

を実行すると 6 ではなく 10 と表示する。これは

```
void func(int a)
{
    a = 6;
}
```

が値渡しになっていて、func() の中で a の値を変えても、元の n の値は変わりません。これを

```
#include <iostream>

using namespace std;

void func(int *a)
{
    *a = 6;
}

int main()
{
    int n = 10;
    func(&n);
    cout << n << endl;
    getchar();
}
```

とポインタを使って、アドレス渡しにすると 6 と表示します。
　旺文社数学Aには次のプログラムがある。

```
10 REM 素数の発見
20 INPUT "求める素数の個数";X
30 DIM P(X)
40 J=1:P(J)=2
50 PRINT J,P(J)
60 N=3
70 FOR M=1 TO J
80    IF INT(N/P(M))*P(M)=N THEN 130
90 NEXT M
100 J=J+1
110 P(J)=N
120 PRINT J,P(J)
130 N=N+1
140 IF J<X THEN 70
150 END
```

30 行目の DIM P(X) が配列の宣言で P(0) から P(X) までの X+1 個の実数型の配列を動的に宣言している。この BASIC のプログラムは GOTO 文があるので理解するのが難しいが

1. 求める素数の個数 X を入力する。
2. 大きさ X の配列 P(X) を確保する。　BASIC の配列は P(1), P(2), $\cdots$. P(X) です
3. J = 1 とし、P(J) = 2 とする。(最初の素数 2 を P(1) に保存する)
4. J と P(J) を表示する
5. N = 3 とする。次の素数の候補です。
6. すでに発見された素数で N が割り切れるか調べ割り切れればステップ 9 へ
7. N は素数なので J を一つ増やし、P(J) に保存する
8. J と P(J) を表示する
9. N を一つ増やす
10. 素数が X 個見つかっていなければ、ステップ 6 へ

ということをしています。C++ 用のもっと分かりやすいアルゴリズムは

1. 求める素数の個数 X を入力する。
2. 大きさ X の配列 P[X] を確保する。
3. J = 0 とし、P[J] = 2 とする。(最初の素数 2 を P[0] に保存する)
4. J と P[J] を表示する

5. N = 3 とする。次の素数の候補です。

6. 素数が X 個見つかるまで次を繰り返す。

   (a) 合成数であったかどうかを調べるための変数 FLAG を眞にセット

   (b) すでに発見された素数で N が割り切れるか順に調べ割り切れれば FLAG に偽をセット

   (c) FLAG が眞であれば N は素数なので J を一つ増やし、P[J] に保存し、J と P[J] を表示する

   (d) N を一つ増やす

この問題の厄介なところは、求める素数の個数 X を入力した後、必要な配列の大きさが決まることです。C や C++ でも可能ですが、理解できるように説明するのが厄介です。このような場合、C++ では vector を使うと楽です。

vector を使ってこのアルゴリズムを実現してみましょう。雛形

```
#include <iostream>
using namespace std;

int main()
{
    getchar();
    return 0;
}
```

から始めます。

求める素数の個数 X を入力する。

を組み込みます。

```
#include <iostream>
using namespace std;

int main()
{
    int x;

    cout << "求める素数の個数："; cin >> x;
    getchar();
    return 0;
}
```

となります。

大きさ X の配列 P[X] を確保する。ですが、ここでは配列ではなく vector を使います。

vector を使うためにはまず

```
#include <vector>
```

とします。vector を使うために必要な宣言を読み込みます。

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int x;

    cout << "求める素数の個数："; cin >> x;
    getchar();
    return 0;
}
```

となります。vector の大きさをここで決めるのではなく、Python でリストを使った時のように、空の vector を作り、素数が見つかる度に、vector に保存していきます。空の vector を宣言します。

```
    vector<int> prime;
```

とします。これで int の値を保存する名前 prime という vector を確保します。prime には何も保存されていません。これを組み込みます。

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数："; cin >> x;
    getchar();
    return 0;
}
```

J = 0 とし、P[J] = 2 とする。(最初の素数 2 を P[0] に保存する) は

```
   prime.push_back(2);
```

で良いです、Python は append() でしたが、vector は `push_back()` を使います。何個の素数を保存したか保持している変数 J は必要ないです。、prime.size() で保存した素数の個数が分かります。これを組み込みます。

```
#include <iostream>
#include <vector>
using namespace std;
```

```
int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数："; cin >> x;
    prime.push_back(2);
    getchar();
    return 0;
}
```

J と P[J] を表示する
は

```
    cout << prime.size() << " : " << prime[prime.size()-1] << endl;
```

で良いです。見つかった素数の個数は prime.size() で、最後に見つけた素数は prime[prime.size()-1] です。prime[index] で index+1 番目の素数が分かります。index の動く範囲は $0 \leq index \leq prime.size() - 1$ です。これを組み込みます。

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数："; cin >> x;
    prime.push_back(2);
    cout << prime.size() << " : " << prime[prime.size()-1] << endl;
    getchar();
    return 0;
}
```

N = 3 とする。次の素数の候補です。は

```
    int n = 3;
```

で良いです。これを組み込みます。

```
#include <iostream>
#include <vector>
using namespace std;
```

```
int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数："; cin >> x;
    prime.push_back(2);
    cout << prime.size() << " : " << prime[prime.size()-1] << endl;
    int n = 3;
    getchar();
    return 0;
}
```

素数が X 個見つかるまで次を繰り返す。は while 文を使って

```
    while (prime.size() < x) {
    }
```

で良いです。これを組み込みます。

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数："; cin >> x;
    prime.push_back(2);
    cout << prime.size() << " : " << prime[prime.size()-1] << endl;
    int n = 3;
    while (prime.size() < x) {
    }
    getchar();
    return 0;
}
```

合成数であったかどうかを調べるための変数 FLAG を眞にセットは bool の値を保持する変数 flag を使って

```
        bool flag = true;
```

で良いです。これを組み込みます。

```
#include <iostream>
```

```
#include <vector>
using namespace std;

int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数："; cin >> x;
    prime.push_back(2);
    cout << prime.size() << " : " << prime[prime.size()-1] << endl;
    int n = 3;
    while (prime.size() < x) {
        bool flag = true;
    }
    getchar();
    return 0;
}
```

すでに発見された素数で N が割り切れるか順に調べ割り切れれば FLAG に偽をセットは for 文を使って

```
        for (int i=0; i<prime.size(); i++) {
            if (n % prime[i] == 0) {
                flag = false;
                break;
            }
        }
```

で良いです。これを組み込みます。

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数："; cin >> x;
    prime.push_back(2);
    cout << prime.size() << " : " << prime[prime.size()-1] << endl;
    int n = 3;
    while (prime.size() < x) {
```

```
        bool flag = true;
        for (int i=0; i<prime.size(); i++) {
            if (n % prime[i] == 0) {
                flag = false;
                break;
            }
        }
    }
    getchar();
    return 0;
}
```

FLAG が眞であれば N は素数なので J を一つ増やし、P[J] に保存し、J と P[J] を表示するは if 文を使って

```
        if (flag == true) {
            prime.push_back(n);
            cout << prime.size() << " : " << prime[prime.size()-1] << endl;
        }
```

で良いです。これを組み込みます。

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数 : "; cin >> x;
    prime.push_back(2);
    cout << prime.size() << " : " << prime[prime.size()-1] << endl;
    int n = 3;
    while (prime.size() < x) {
        bool flag = true;
        for (int i=0; i<prime.size(); i++) {
            if (n % prime[i] == 0) {
                flag = false;
                break;
            }
        }
        if (flag == true) {
            prime.push_back(n);
```

```
            cout << prime.size() << " : " << prime[prime.size()-1] << endl;
        }
    }
    getchar();
    return 0;
}
```

N を一つ増やすは

```
        n++;
```

で良いです。これを組み込みます。

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数 : "; cin >> x;
    prime.push_back(2);
    cout << prime.size() << " : " << prime[prime.size()-1] << endl;
    int n = 3;
    while (prime.size() < x) {
        bool flag = true;
        for (int i=0; i<prime.size(); i++) {
            if (n % prime[i] == 0) {
                flag = false;
                break;
            }
        }
        if (flag == true) {
            prime.push_back(n);
            cout << prime.size() << " : " << prime[prime.size()-1] << endl;
        }
        n++;
    }
    getchar();
    return 0;
}
```

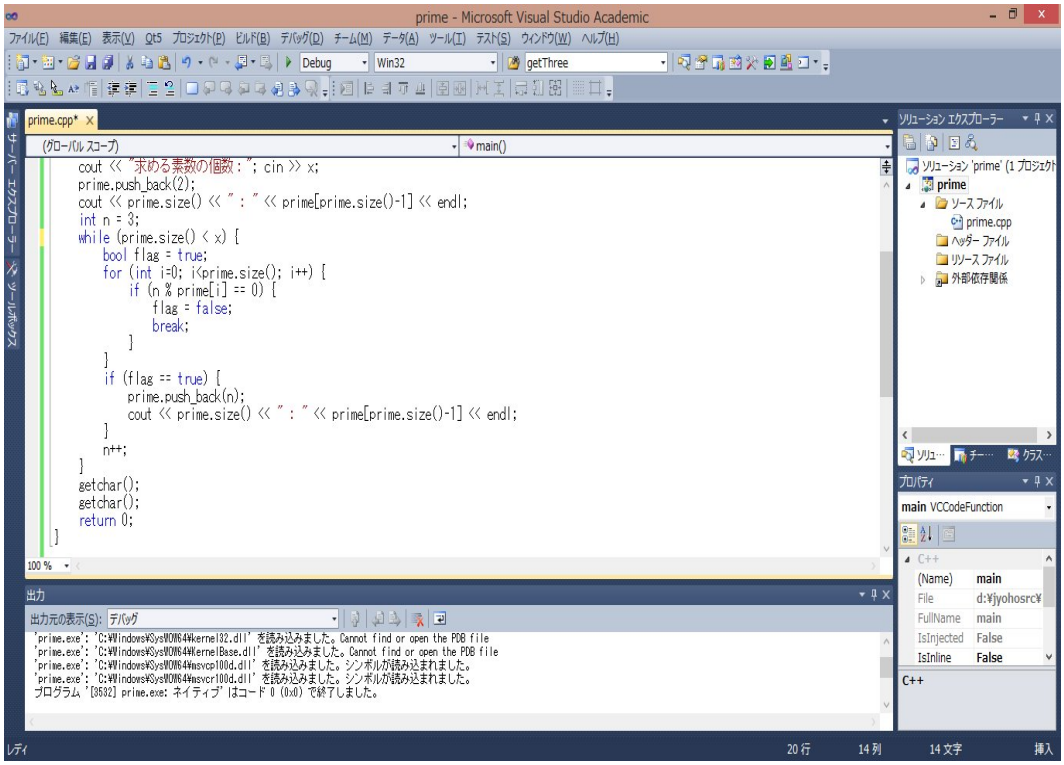
最後に cin を使ったので

```
    getchar();
```

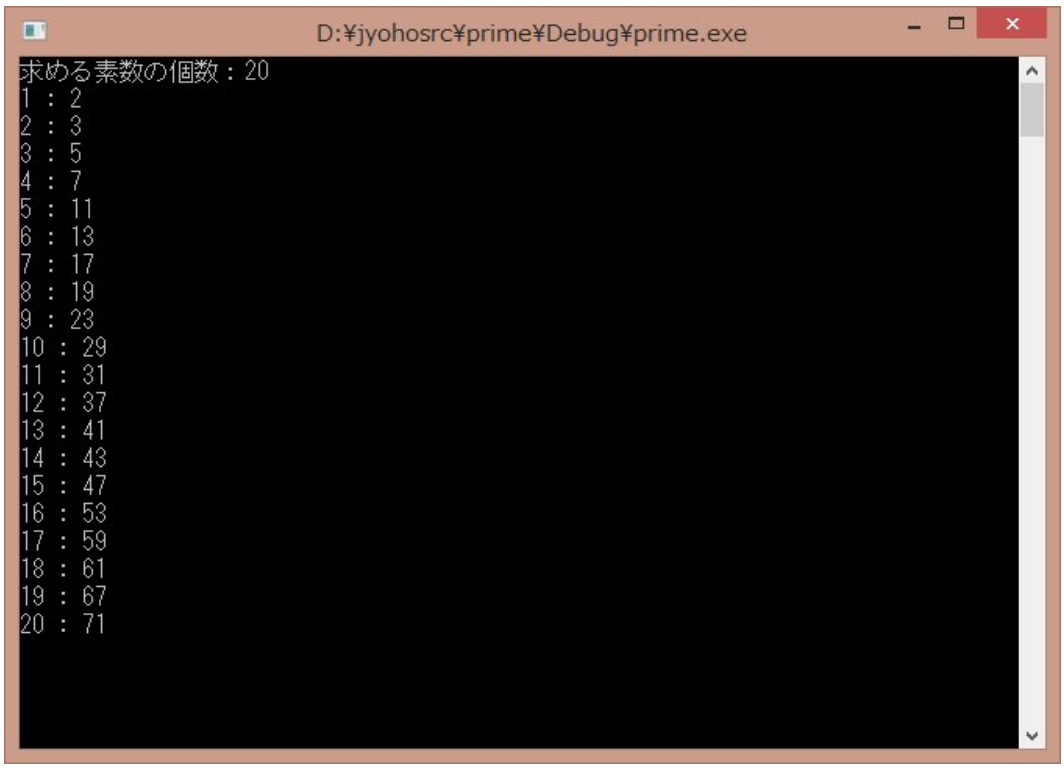
を追加します。

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数："; cin >> x;
    prime.push_back(2);
    cout << prime.size() << " : " << prime[prime.size()-1] << endl;
    int n = 3;
    while (prime.size() < x) {
        bool flag = true;
        for (int i=0; i<prime.size(); i++) {
            if (n % prime[i] == 0) {
                flag = false;
                break;
            }
        }
        if (flag == true) {
            prime.push_back(n);
            cout << prime.size() << " : " << prime[prime.size()-1] << endl;
        }
        n++;
    }
    getchar();
    getchar();
    return 0;
}
```

完成しました。



実行します。



プログラムを更に手直しします。表示を綺麗にしてみましょう。

```
#include <iomanip>
```

を追加します。

```
#include <iostream>
#include <vector>
#include <iomanip>
using namespace std;

int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数："; cin >> x;
    prime.push_back(2);
    cout << prime.size() << " : " << prime[prime.size()-1] << endl;
    int n = 3;
    while (prime.size() < x) {
        bool flag = true;
        for (int i=0; i<prime.size(); i++) {
            if (n % prime[i] == 0) {
                flag = false;
                break;
            }
        }
        if (flag == true) {
            prime.push_back(n);
            cout << prime.size() << " : " << prime[prime.size()-1] << endl;
        }
        n++;
    }
    getchar();
    getchar();
    return 0;
}
```

二ヶ所にある

```
    cout << prime.size() << " : " << prime[prime.size()-1] << endl;
```

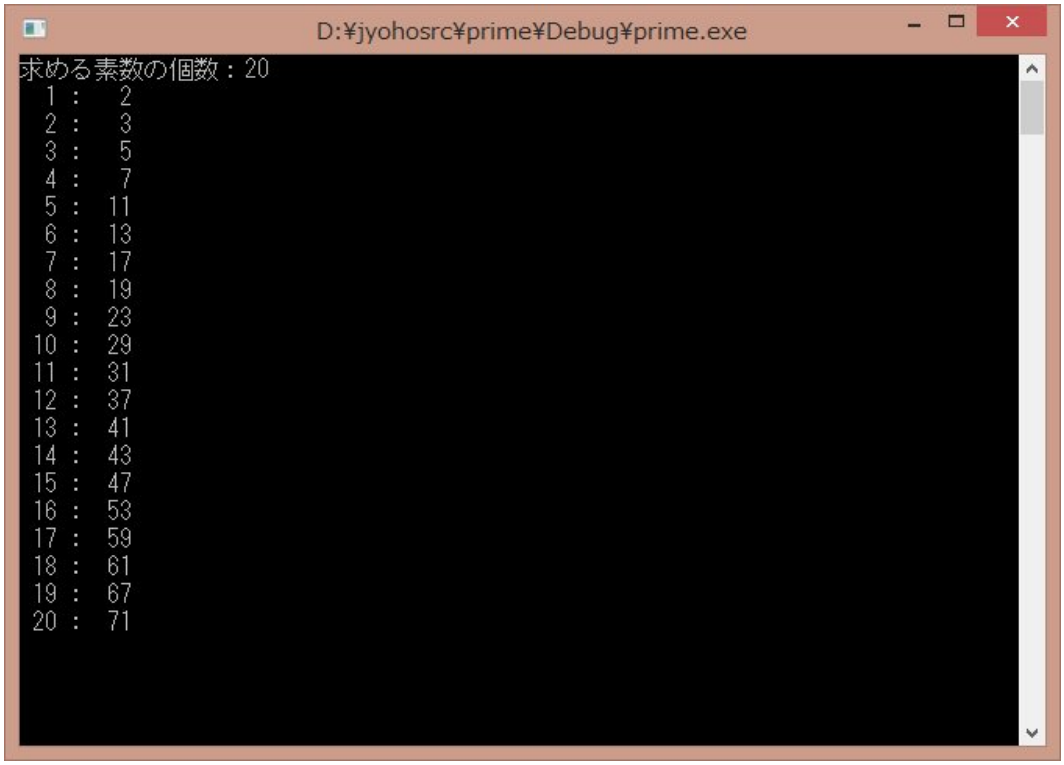
を

```
    cout <<setw(3) <<  prime.size() << " : " <<setw(3) << prime[prime.size()-1] << endl;
```

に変えます。

```
#include <iostream>
#include <vector>
#include <iomanip>
using namespace std;

int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数："; cin >> x;
    prime.push_back(2);
    cout << setw(3) << prime.size() << " : " << setw(3) << prime[prime.size()-1] << endl;
    int n = 3;
    while (prime.size() < x) {
        bool flag = true;
        for (int i=0; i<prime.size(); i++) {
            if (n % prime[i] == 0) {
                flag = false;
                break;
            }
        }
        if (flag == true) {
            prime.push_back(n);
            cout << setw(3) << prime.size() << " : " << setw(3) <<
                     prime[prime.size()-1] << endl;
        }
        n++;
    }
    getchar();
    getchar();
    return 0;
}
```

実行します。

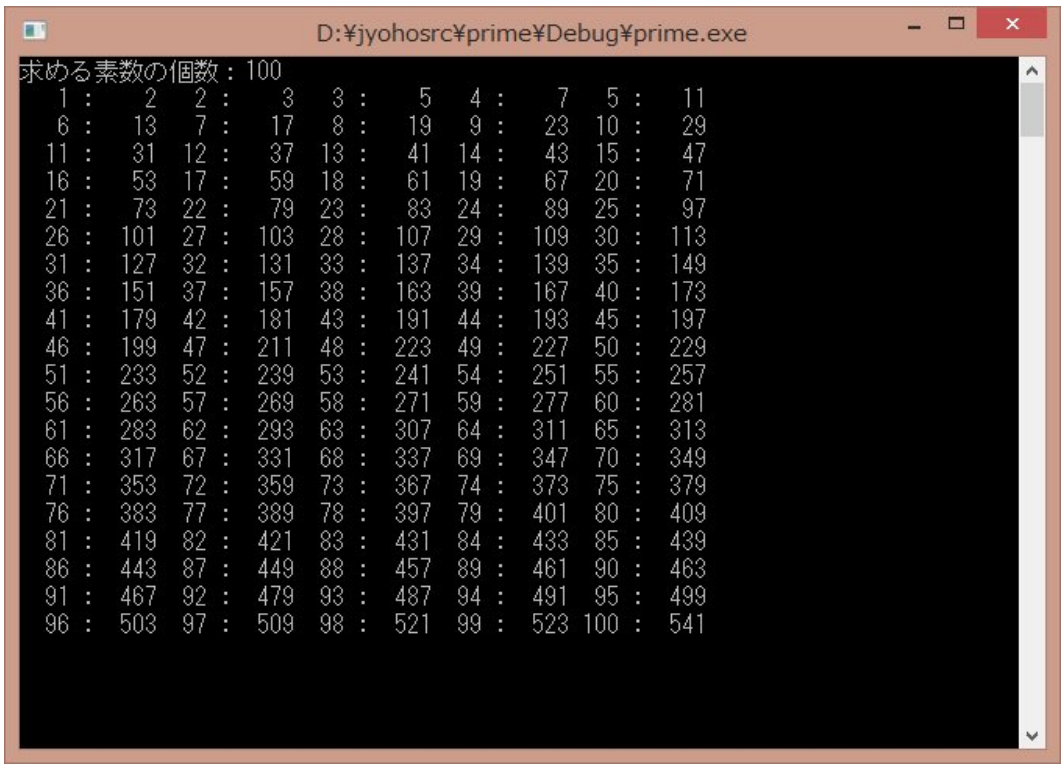


更に、一行に 5 個づつ表示するようにします。

```
#include <iostream>
#include <vector>
#include <iomanip>
using namespace std;

int main()
{
    int x;
    vector<int> prime;

    cout << "求める素数の個数："; cin >> x;
    prime.push_back(2);
    cout << setw(4) << prime.size() << " : " << setw(4) << prime[prime.size()-1]l;
    int n = 3;
    while (prime.size() < x) {
        bool flag = true;
        for (int i=0; i<prime.size(); i++) {
            if (n % prime[i] == 0) {
                flag = false;
                break;
            }
```

```
        }
        if (flag == true) {
            prime.push_back(n);
            cout << setw(4) << prime.size() << " : " << setw(4) <<
                    prime[prime.size()-1];
            if ((prime.size() - 1)  % 5 == 4) cout << endl;
        }
        n++;
    }
    getchar();
    getchar();
    return 0;
}
```

と修正します。実行します。



`cout << setw(3) << prime.size() << " : " << setw(3) << prime[prime.size()-1] << endll;`

で setw(3) は三桁取って次の整数値を表示することを指示している。setw(3) を使うために

`#include <iomanip>`

が必要です。

`cout << setw(4) << prime.size() << " : " << setw(4) << prime[prime.size()-1]l;`

の代わりに C の関数を使って

```
printf("%4d : %4d  ", prime.size(), prime[prime.size()-1]);
```

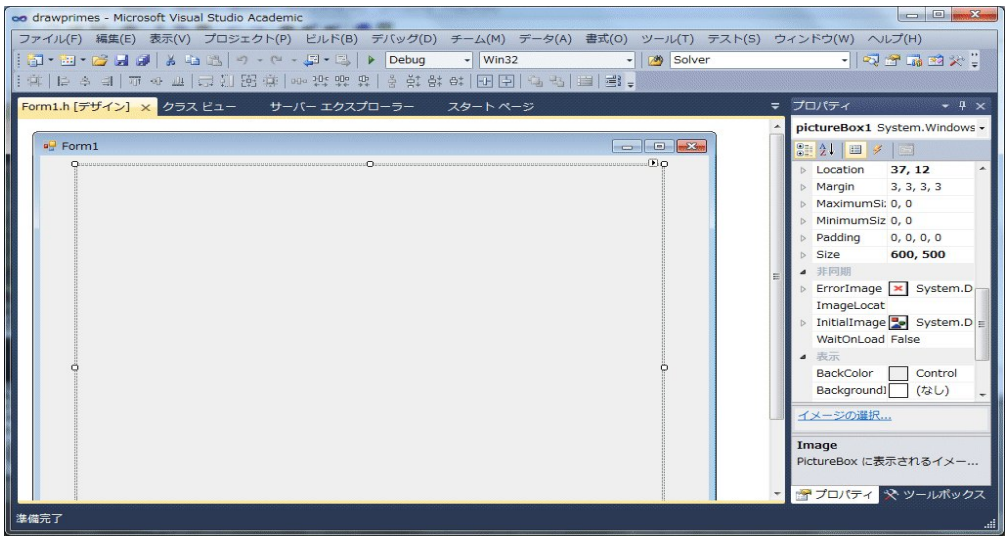
で、表示しても良いです。prime.size() と prime[prime.size()-1] を整数値として、四桁使って表示します。

```
cout << setw(4) << prime.size() << " : " << setw(4) <<
        prime[prime.size()-1];
if ((prime.size() - 1)  % 5 == 4) cout << endl;
```

は、五個づつ表示するために、改行を出力させています。

素数の現れる様子をグラフにしてみましょう。

Windows フォーム・アプリケーションで、プロパティで Form1 を Size : 700,550 にし、ツールボックスから PictureBox を選び、Form1 の中央に PictureBox を配置する。 Size : 600,500 にする。

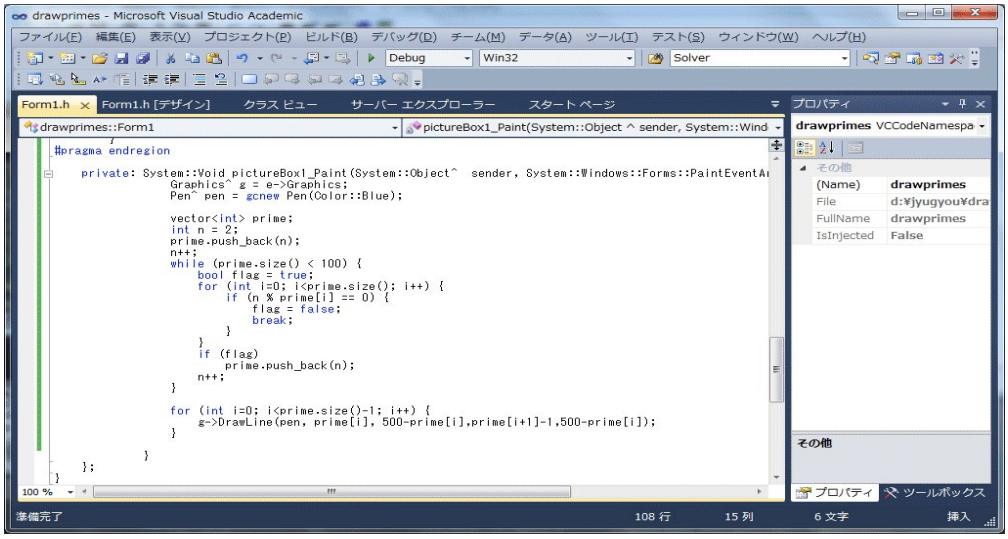


PictureBox をクリックし、プロパティのイベントで Paint をダブルクリックする。

```
  private: System::Void pictureBox1_Paint(System::Object^  sender,
System::Windows::Forms::PaintEventArgs^  e) {
        Graphics^ g = e->Graphics;
        Pen^ pen = gcnew Pen(Color::Blue);

        vector<int> prime;
        int n = 2;
        prime.push_back(n);
        n++;
        while (prime.size() < 100) {
            bool flag = true;
            for (int i=0; i<prime.size(); i++) {
                if (n % prime[i] == 0) {
                    flag = false;
```

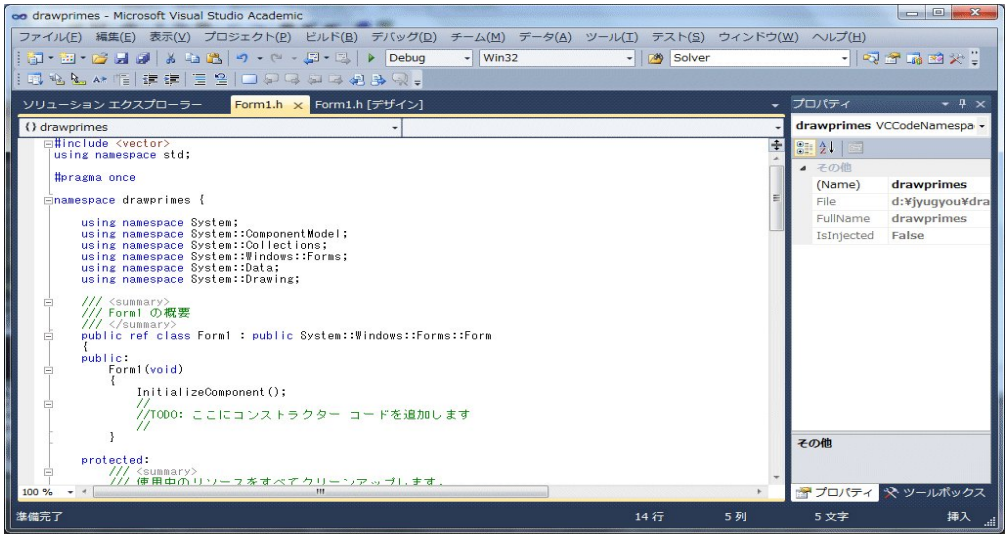
```
                    break;
                }
            }
            if (flag)
                prime.push_back(n);
            n++;
        }

        for (int i=0; i<prime.size()-1; i++) {
            g->DrawLine(pen, prime[i], 500-prime[i],prime[i+1]-1,500-prime[i]);
        }
    }
```

となるように、打ち込みます。

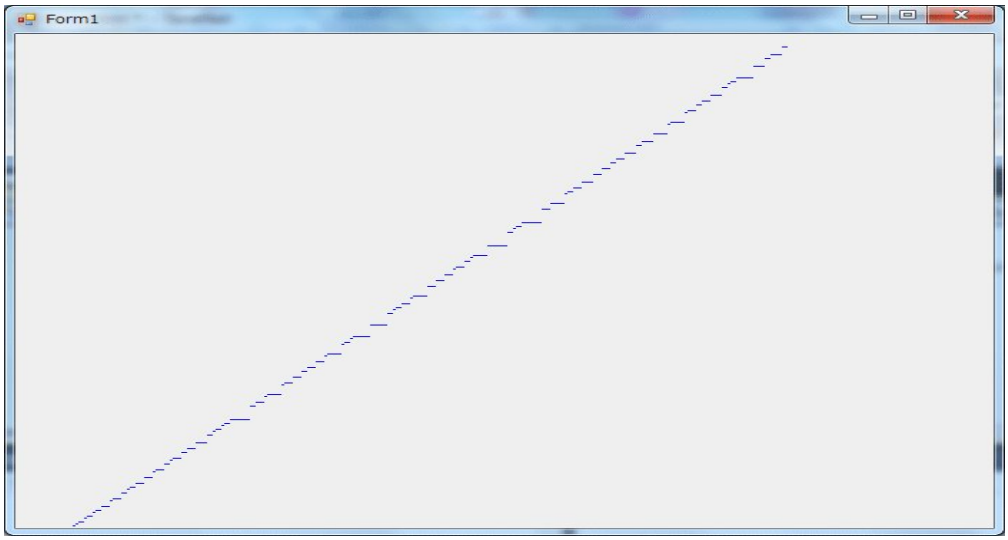



```
#include <vector>
using namespace std;
```

を Form1.h の先頭に打ち込む。

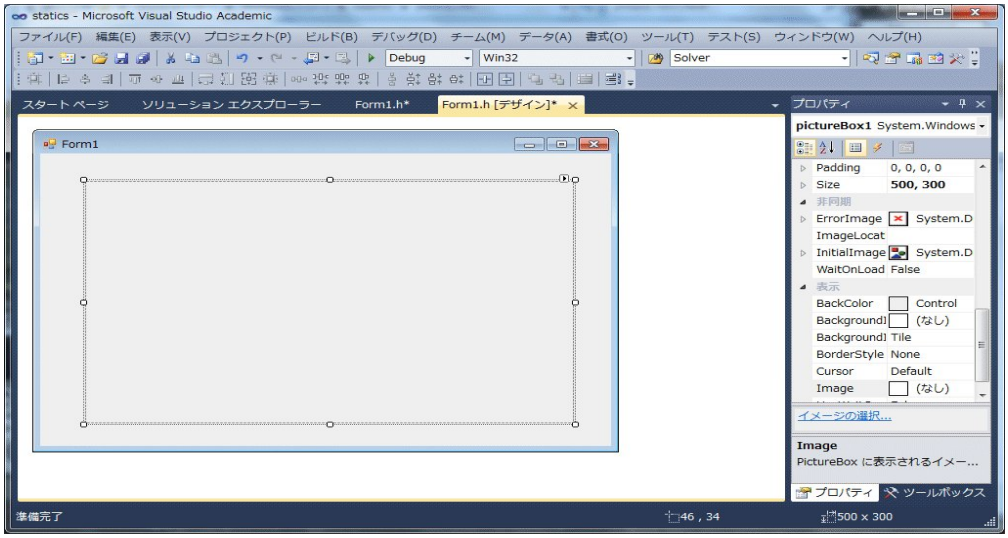


実行する。



最初の１０００個の素数の出現間隔をグラフにしてみよう。すなわち、上図の横線の長さの個数をヒストグラムにしてみよう。

Windows フォーム・アプリケーションで、プロパティで Form1 を Size：600,400 にし、ツールボックスから PictureBox を選び、Form1 の中央に PictureBox を配置する。 Size：500,300 にする。



PictureBox をクリックし、プロパティのイベントで Paint をダブルクリックする。

```
#include <vector>
using namespace std;
```

を Form1.h の先頭に打ち込む。

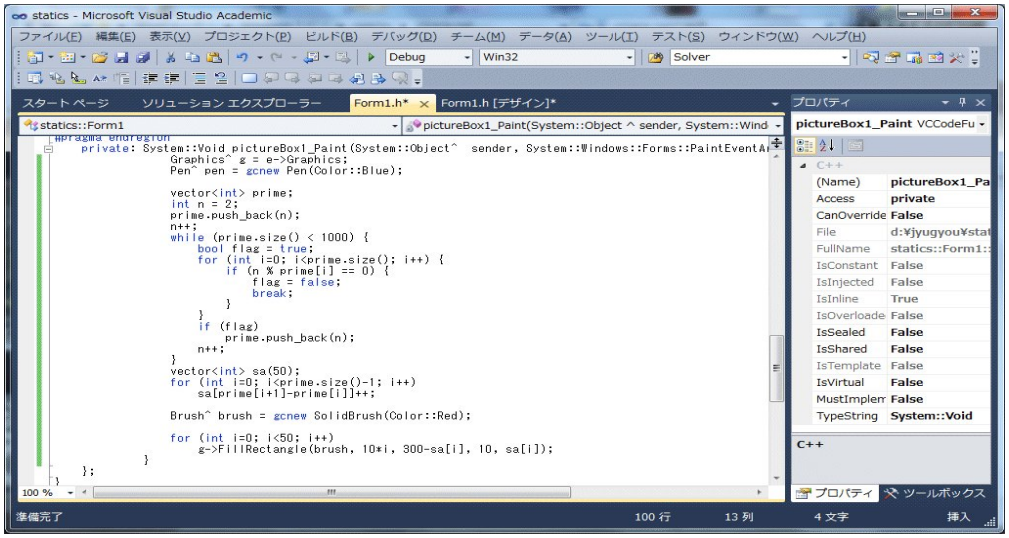
```
    private: System::Void pictureBox1_Paint(System::Object^  sender,
 System::Windows::Forms::PaintEventArgs^  e) {
    }
```

を

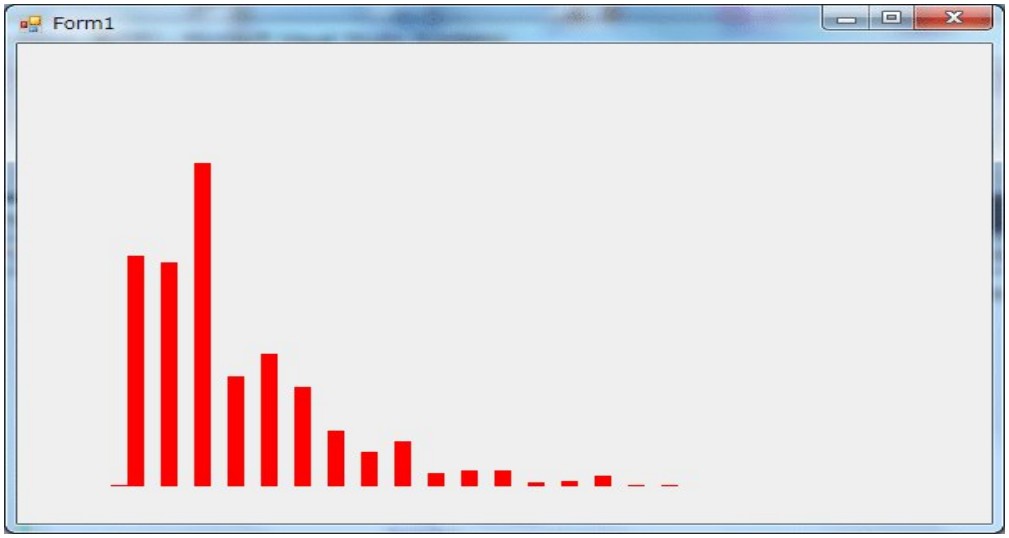
```
    private: System::Void pictureBox1_Paint(System::Object^  sender,
 System::Windows::Forms::PaintEventArgs^  e) {
        Graphics^ g = e->Graphics;
        Pen^ pen = gcnew Pen(Color::Blue);

        vector<int> prime;
        int n = 2;
        prime.push_back(n);
        n++;
        while (prime.size() < 1000) {
            bool flag = true;
            for (int i=0; i<prime.size(); i++) {
                if (n % prime[i] == 0) {
                    flag = false;
                    break;
                }
            }
            if (flag)
```

```
            prime.push_back(n);
        n++;
    }
    vector<int> sa(50);
    for (int i=0; i<prime.size()-1; i++)
        sa[prime[i+1]-prime[i]]++;

    Brush^ brush = gcnew SolidBrush(Color::Red);

    for (int i=0; i<50; i++)
        g->FillRectangle(brush, 10*i, 300-sa[i], 10, sa[i]);
}
```

とする。



実行する



```
vector<int> sa(50);
for (int i=0; i<prime.size()-1; i++)
    sa[prime[i+1]-prime[i]]++;

Brush^ brush = gcnew SolidBrush(Color::Red);

for (int i=0; i<50; i++)
    g->FillRectangle(brush, 10*i, 300-sa[i], 10, sa[i]);
```

の

```
vector<int> sa(50);
```

は、整数値を保持するベクトル sa を５０個 sa[0],sa[1],sa[2], ... , sa[49] 予約している。このように宣言すると `sa.push_back(i)` のように整数値を追加して行かなくても sa[0],sa[1],sa[2], ... , sa[49] が 0 に初期化されていて、そのまま値を代入したり、参照できる。

```
for (int i=0; i<prime.size()-1; i++)
    sa[prime[i+1]-prime[i]]++;
```

で、sa[] の prime[i+1]-prime[i] 番目 (sa[prime[i+1]-prime[i]]) を一つ増やしている。これで sa[1] に間隔が 1 (例えば 2 と 3）の連続する素数の組の個数、 sa[2] に間隔が 2 (例えば 3 と 5 や 5 と 7）の連続する素数の組の個数がセットされることになる。

```
Brush^ brush = gcnew SolidBrush(Color::Red);

for (int i=0; i<50; i++)
    g->FillRectangle(brush, 10*i, 300-sa[i], 10, sa[i]);
```

で、ヒストグラムを表示している。

更に、平均と標準偏差を計算し、表示してみよう。

```
double s = 0;
for (int i=0; i<sa.size(); i++)
    s += i*sa[i];
double heikin = s / (prime.size()-1);
double bunsan = 0;
for (int i=0; i<sa.size(); i++)
    bunsan += (i-heikin)*(i-heikin)*sa[i];
bunsan /= (prime.size()-1);
double hyoujyunhensa = sqrt(bunsan);

System::Drawing::Font^ drawFont = gcnew System::Drawing::Font("Arial", 16);
Brush^ drawBrush = gcnew SolidBrush(Color::Black);
System::String^ drawString = "平均 = ";
drawString += System::Convert::ToString(heikin);
g->DrawString(drawString, drawFont, drawBrush, 150, 50);
```

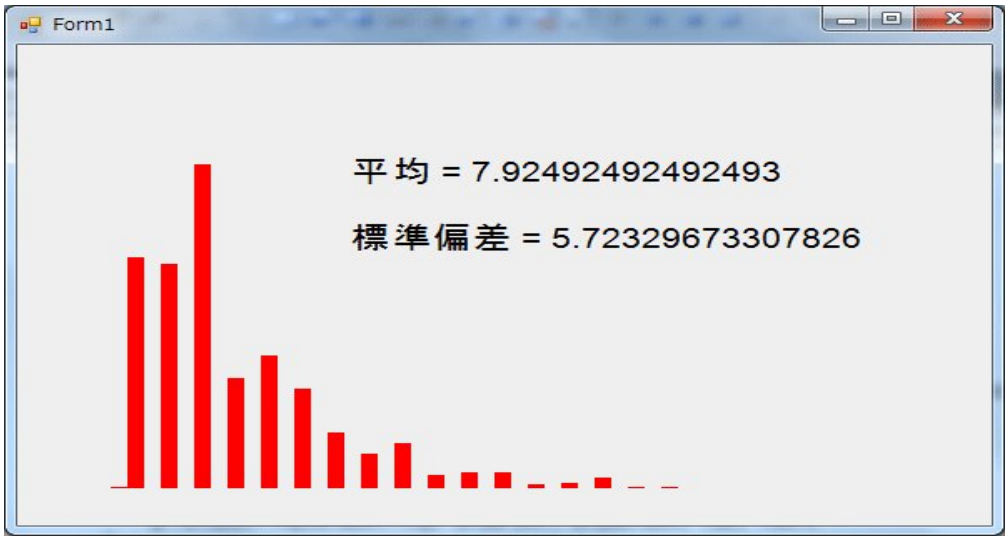
```
        drawString = "標準偏差 = ";
        drawString += System::Convert::ToString(hyoujyunhensa);
        g->DrawString(drawString, drawFont, drawBrush, 150, 100);
```

を追加する。結局、

```
    private: System::Void pictureBox1_Paint(System::Object^  sender,
 System::Windows::Forms::PaintEventArgs^  e) {
        Graphics^ g = e->Graphics;
        Pen^ pen = gcnew Pen(Color::Blue);

        vector<int> prime;
        int n = 2;
        prime.push_back(n);
        n++;
        while (prime.size() < 1000) {
            bool flag = true;
            for (int i=0; i<prime.size(); i++) {
                if (n % prime[i] == 0) {
                    flag = false;
                    break;
                }
            }
            if (flag)
                prime.push_back(n);
            n++;
        }
        vector<int> sa(50);
        for (int i=0; i<prime.size()-1; i++)
            sa[prime[i+1]-prime[i]]++;

        Brush^ brush = gcnew SolidBrush(Color::Red);

        for (int i=0; i<50; i++)
            g->FillRectangle(brush, 10*i, 300-sa[i], 10, sa[i]);

        double s = 0;
        for (int i=0; i<sa.size(); i++)
            s += i*sa[i];
        double heikin = s / (prime.size()-1);
        double bunsan = 0;
        for (int i=0; i<sa.size(); i++)
            bunsan += (i-heikin)*(i-heikin)*sa[i];
        bunsan /= (prime.size()-1);
```

```
        double hyoujyunhensa = sqrt(bunsan);

        System::Drawing::Font^ drawFont = gcnew System::Drawing::Font("Arial", 16);
        Brush^ drawBrush = gcnew SolidBrush(Color::Black);
        System::String^ drawString = "平均 = ";
        drawString += System::Convert::ToString(heikin);
        g->DrawString(drawString, drawFont, drawBrush, 150, 50);
        drawString = "標準偏差 = ";
        drawString += System::Convert::ToString(hyoujyunhensa);
        g->DrawString(drawString, drawFont, drawBrush, 150, 100);
    }
```

となった。実行すると



となる。

```
        double s = 0;
        for (int i=0; i<sa.size(); i++)
            s += i*sa[i];
        double heikin = s / (prime.size()-1);
        double bunsan = 0;
        for (int i=0; i<sa.size(); i++)
            bunsan += (i-heikin)*(i-heikin)*sa[i];
        bunsan /= (prime.size()-1);
        double hyoujyunhensa = sqrt(bunsan);
```

の部分で、平均と分散と標準偏差を計算している。これは高等学校の統計で学ぶ内容である。

```
        System::Drawing::Font^ drawFont = gcnew System::Drawing::Font("Arial", 16);
        Brush^ drawBrush = gcnew SolidBrush(Color::Black);
        System::String^ drawString = "平均 = ";
        drawString += System::Convert::ToString(heikin);
```

```
g->DrawString(drawString, drawFont, drawBrush, 150, 50);
drawString = "標準偏差 = ";
drawString += System::Convert::ToString(hyoujyunhensa);
g->DrawString(drawString, drawFont, drawBrush, 150, 100);
```

の部分で、平均と標準偏差を表示している。文字列を表示するには DrawString() を使う。表示文字列とフォントとブラシと表示位置を指定する。

```
System::Drawing::Font^ drawFont = gcnew System::Drawing::Font("Arial", 16);
```

は、フォント drawFont を定義している。16 はフォントの大きさである。

```
Brush^ drawBrush = gcnew SolidBrush(Color::Black);
```

は、ブラシ drawBrush を定義している。

```
System::String^ drawString = "平均 = ";
```

は、文字列 drawString を定義している。これは VC++ 独特の定義である (C++ の範囲外)。こうするものだと思うしかない。

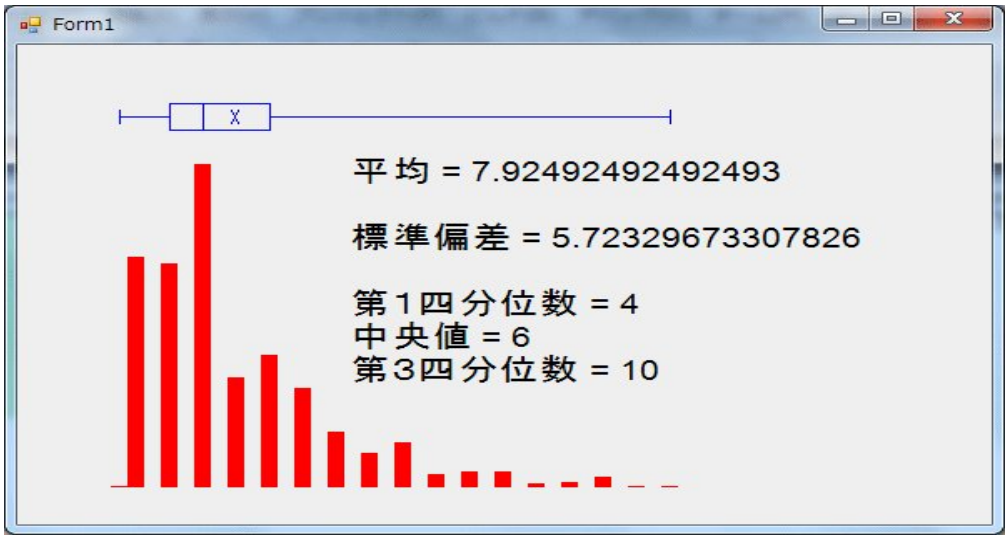
```
drawString += System::Convert::ToString(heikin);
```

は、実数値 heikin を文字列に変換し、”平均 = ” に連接している。

```
g->DrawString(drawString, drawFont, drawBrush, 150, 50);
```

で、文字列の左上隅が (150,50) の位置に drawString の内容を表示している。”標準偏差 = ” の表示も同様である。VC++ を使うときは、このようにするものだと理屈抜きに覚えるしかない。

問題：箱ひげ図も表示するように修正しなさい。



例題：配列の要素を並べ替える（ソートと言います）プログラムを作れ。次は入力した数値を大きい順に並べ替える BASIC のプログラムである。

```
10 REM データの並べ替え
20 INPUT "データの個数";N
30 DIM A(N)
40 FOR J=1 TO N
50     INPUT A(J)
60 NEXT J
70 FOR J=1 TO N-1
80     FOR K=J+1 TO N
90         IF A(J) > A(K) THEN 110
100        X=A(J):A(J)=A(K):A(K)=X
110    NEXT K
120    PRINT A(J)
130 NEXT J
140 PRINT A(N)
150 END
```

これは最小値選択法と呼ばれるアルゴリズムです。データの個数が多くなると色々なアルゴリズムを使う必要があります。この分野は最も研究されている分野の 1 つです。

これを C に翻訳すると次のようになる。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    double *a, x;
    int n, j, k;

    printf("データの個数"); scanf("%d", &n);
    if ((a = (double *)malloc((n+1)*sizeof(double))) == NULL) {
        printf("malloc error!\n");
        exit(1);
    }
    for (j=1; j<=n; i++)
        scanf("%lf", &a[j]);
    for (j=1; j<n; j++) {
        for (k=j+1; k<=n; k++) {
            if (a[j]>a[k]) continue;
            x=a[j]; a[j]=a[k]; a[k]=x;
        }
        printf("%f\n", a[j]);
    }
    printf("%f\n", a[n]);
    free(a);
    getchar();
```

```
    return 0;
}
```

continue 文が使われていることに注意。

```
        for (k=j+1; k<=n; k++) {
            if (a[j]>a[k]) continue;
            x=a[j]; a[j]=a[k]; a[k]=x;
        }
```

で、if 文で $a[j] > a[k]$ であれば、以下の命令達を無視し、すぐ for 文の k++ を実行します。
C++ だと次のようになります。

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

using namespace std;

int main(int argc, char* argv[])
{
    int n;
    double *a;

    cout << "データの個数 N="; cin >> n;
    a = new double[n];
    for (int i=0; i<n; i++) {
        cout << "a[" << i+1 << "]="; cin >> a[i];
    }
    getchar();
    for (int i=0; i<n; i++)
        for (int k=i+1; k<n; k++)
            if (a[i] > a[k]) {
                double temp = a[i];
                a[i] = a[k];
                a[k] = temp;
            }
    for (int i=0; i<n; i++) {
        cout << a[i] << '\t';
        if ((i+1) % 5 == 0) cout << endl;
    }
    delete[] a:
    getchar();
    return 0;
}
```

C では実行時に配列の大きさを決めることができないのでポインタを宣言し（今の場合 double *a)、malloc() 関数を用いて必要な領域を割り当てましたが、C++ では

```
double *a;
```

とポインタを宣言し、new 命令を用いて

```
a = new double[n];
```

で、必要な領域を割り当てます。領域が不要になれば

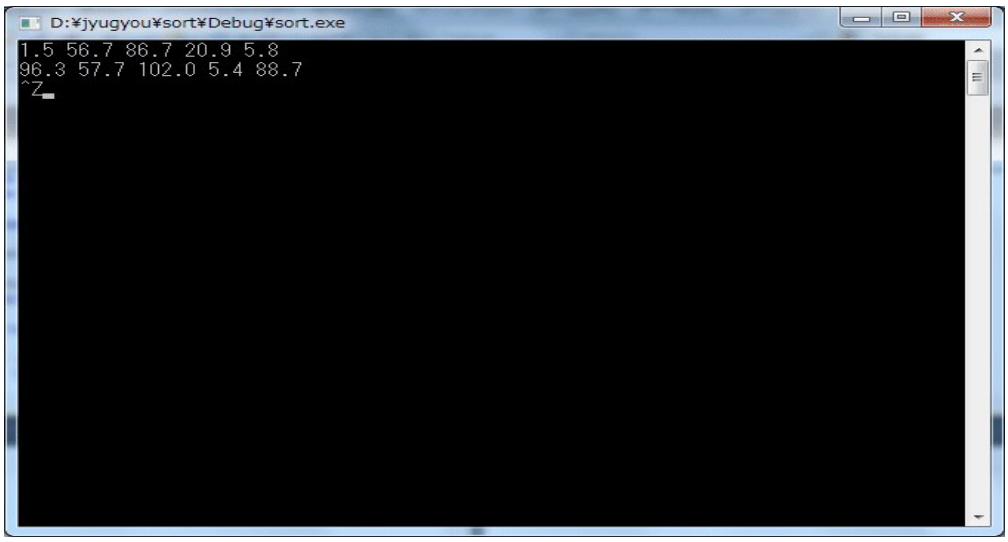
```
delete[] a;
```

で、領域を開放します。そうすれば配列と同様な使い方ができる。

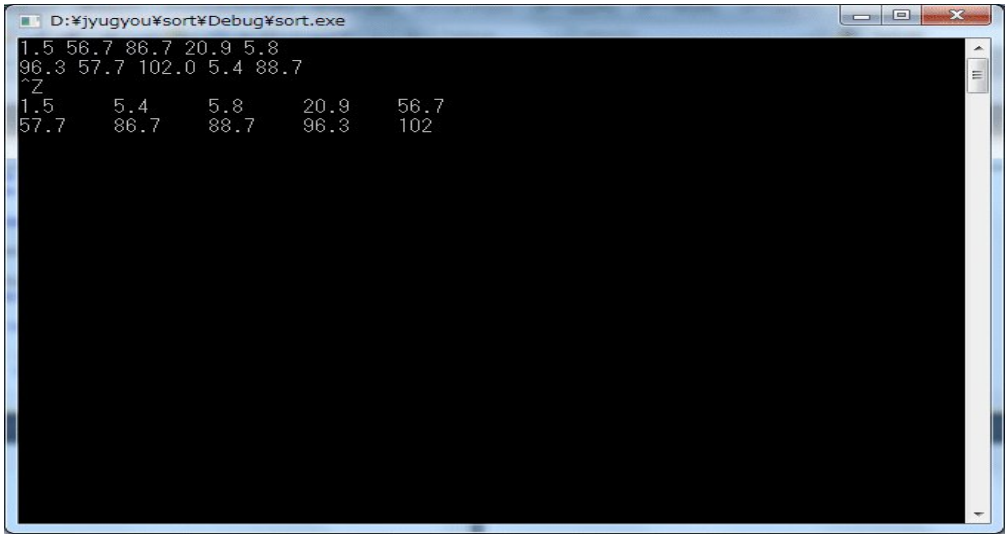
現在は C++ では、配列の代わりに、もっと便利なコンテナクラス vector が使えます。vector を使ったプログラムの例は次のようになります。

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<double> p;
    double data;

    while (cin >> data) p.push_back(data);

    for (int i=1; i<p.size(); i++) {
        double val = p[i];
        for (int j=i-1; j>=0; j--) {
            if (val < p[j]) {
                p[j+1] = p[j];
                p[j] = val;
            } else {
                break;
            }
        }
    }
    for (int i=0; i<p.size(); i++) {
        cout << p[i] << '\t';
        if ((i+1) % 5 == 0) cout << endl;
    }
    getchar();
    return 0;
}
```

実行する。



データの終わりは Ctrl + Z として、Enter キーを押す。




```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<double> p;
    double data;

    while (cin >> data) p.push_back(data);
```

```
    for (int i=1; i<p.size(); i++) {
        double val = p[i];
        for (int j=i-1; j>=0; j--) {
            if (val < p[j]) {
                p[j+1] = p[j];
                p[j] = val;
            } else {
                break;
            }
        }
    }
    for (int i=0; i<p.size(); i++) {
        cout << p[i] << '\t';
        if ((i+1) % 5 == 0) cout << endl;
    }
    getchar();
    return 0;
}
```

の

```
#include <iostream>
#include <vector>

using namespace std;
```

は、おきまりの呪文です。ベクトルを使うので

```
#include <vector>
```

が必要です。

```
int main()
{
    vector<double> p;
    double data;

    while (cin >> data) p.push_back(data);

    for (int i=1; i<p.size(); i++) {
        double val = p[i];
        for (int j=i-1; j>=0; j--) {
            if (val < p[j]) {
                p[j+1] = p[j];
                p[j] = val;
            } else {
```

```
                break;
            }
        }
    }
    for (int i=0; i<p.size(); i++) {
        cout << p[i] << '\t';
        if ((i+1) % 5 == 0) cout << endl;
    }
    getchar();
    return 0;
}
```

が main() 関数で、

```
    vector<double> p;
```

で、実数のベクトル p を宣言しています。

```
    double data;
```

で、実数値変数 data を宣言しています。次の while 文で使うためです。

```
    while (cin >> data) p.push_back(data);
```

で、ソートする為のデータを入力し、ベクトル p に保存しています。データの終了には Cntl+Z を使います。

```
    for (int i=1; i<p.size(); i++) {
        double val = p[i];
        for (int j=i-1; j>=0; j--) {
            if (val < p[j]) {
                p[j+1] = p[j];
                p[j] = val;
            } else {
                break;
            }
        }
    }
```

の部分で、並べ替えを行っています。小さい順に並べ替えます。考え方は p[0],p[1], ... ,p[i-1] がすでに小さい順に並んでいるとき、p[i] を p[i-1], p[i-2], ... , p[0] の方向に比較し入れ替えていって、p[i] の入るべき位置を見付け p[0], p[1], p[2], ..., p[i-1],p[i] が小さい順に並ぶようにします。これを i = 1 から i = p.size()-1 まですれば、p[0],p[1],p[2], ... , p[p.size()-1] が小さい順に並びます。

```
            if (val < p[j]) {
                p[j+1] = p[j];
                p[j] = val;
            } else {
                break;
            }
```

の if 文は val が p[j] より小さければ、p[j] を p[j+1] に代入し（p[j] を一つずらし）、p[j] に val を代入します。そうでなければ、break 文で中側の for 文を抜けます（val の入れるべき位置が決まったので、部分的な並べ替えが終わりました）。

```
    for (int i=0; i<p.size(); i++) {
        cout << p[i] << '\t';
            if ((i+1) % 5 == 0) cout << endl;
    }
```

で、結果を表示します。

```
    getchar();
    return 0;
```

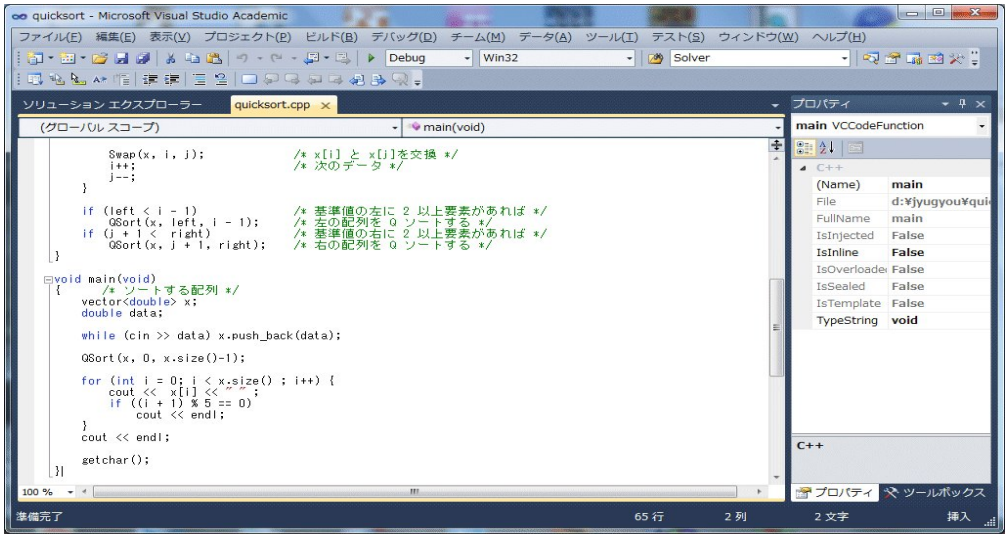
は、ウインドが閉じないように、何らかの入力を待って、修了します。また、int main() としたので、0 を返しています。この値は、Windows では使われないので、何でも良いですが、UNIX の伝統で正しく修了したときは 0 を返します。

クイックソートのプログラムを作ってみよう。

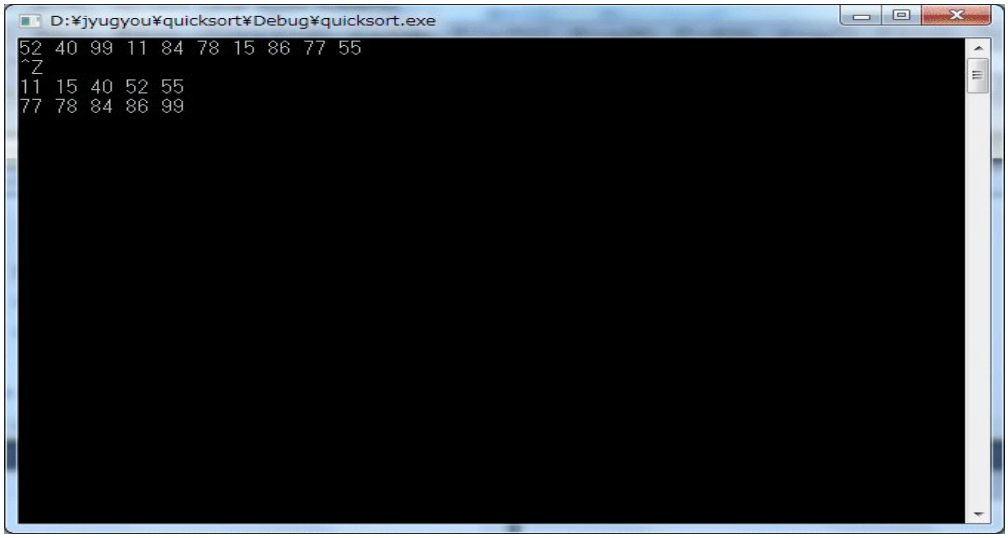
上と同様にして、quicksort.cpp を作る。

```
#include <iostream>
#include <vector>

using namespace std;

  /* 配列の要素を交換する */
void Swap(vector<double>& x, int i, int j)
{
    double temp;

    temp = x[i];
    x[i] = x[j];
    x[j] = temp;
}

  /* クイックソートを行う */
void QSort(vector<double>& x, int left, int right)
{
    int i, j;
    double pivot;

    i = left;                       /* ソートする配列の一番小さい要素の添字 */
    j = right;                      /* ソートする配列の一番大きい要素の添字 */

    pivot = x[(left + right) / 2]; /* 基準値を配列の中央付近にとる */
```

```
    while (true) {                      /* 無限ループ */

        while (x[i] < pivot)        /* pivot より大きい値が */
            i++;                    /* 出るまで i を増加させる */

        while (pivot < x[j])        /* pivot より小さい値が */
            j--;                    /*  出るまで j を減少させる */
        if (i >= j)                 /* i >= j なら */
            break;                  /* 無限ループから抜ける */

        Swap(x, i, j);              /* x[i] と x[j] を交換 */
        i++;                        /* 次のデータ */
        j--;
    }

    if (left < i - 1)               /* 基準値の左に 2 以上要素があれば */
        QSort(x, left, i - 1);      /* 左の配列を Q ソートする */
    if (j + 1 <  right)             /* 基準値の右に 2 以上要素があれば */
        QSort(x, j + 1, right);     /* 右の配列を Q ソートする */
}

void main(void)
{       /* ソートする配列 */
    vector<double> x;
    double data;

    while (cin >> data) x.push_back(data);

    QSort(x, 0, x.size()-1);

    for (int i = 0; i < x.size() ; i++) {
        cout <<  x[i] << " " ;
            if ((i + 1) % 5 == 0)
                cout << endl;
    }
    cout << endl;
    getchar();
}
```



実行する。データの終わりは Ctrl + Z として、Enter キーを押す。



これらのプログラム（インサートソートとクイックソートのプログラム）を Prolog や Scheme のインサートソートとクイックソートのプログラムと見比べて下さい。やっていることは同じですが、そのアルゴリズムの表現方法が言語によって違います。

このプログラムは、分割を Partition() という関数にして、次のようにしても良いです。

```
#include <iostream>
#include <vector>

using namespace std;

int Partition(vector<double>& x, int l, int r)
{
    double v = x[l]; /* 基準値を配列の左端にとる */
    int k = l;
```

```
    int p = r+1;
    double t;

    do k++; while ((x[k] <= v) && (k < r));
    do p--; while (x[p] > v);

    while (k < p) {
        t = x[k]; x[k] = x[p]; x[p] = t;
        do k++; while (x[k] <= v);
        do p--; while (x[p] > v);
    }
    t = x[l]; x[l] = x[p]; x[p] = t;
    return p;
}

void QSort(vector<double>& x, int left, int right)
{
    if (left < right) {
        int q = Partition(x, left, right);
        QSort(x, left, q - 1);
        QSort(x, q + 1, right);
    }
}

void main(void)
{      /* ソートする配列 */
    vector<double> x;
    double data;

    while (cin >> data) x.push_back(data);

    QSort(x, 0, x.size()-1);

    for (int i = 0; i < (int)x.size() ; i++) {
        cout <<  x[i] << " " ;
            if ((i + 1) % 5 == 0)
                cout << endl;
    }
    cout << endl;
    getchar();
}
```

＜補足＞　数の集合の小さい方から k 番目のデータを求めるには、ソート（小さい順に並べ替え）して、 k 番目のデータを求めればいいですが、ソートには時間がかかります。この為のアルゴリズムとして次のようなものが知られています。

１．　データサイズがある一定数 Q よりも少なければデータをソートし、k 番目の要素を返す。多ければ、要素数が Q のチャンクへ分割し、次のステップへ進む。
２．　書くチャンクをソートし、それぞれの中央値を特定する。
３．　再帰的にこのルーチンをコールし、前ステップで得た中央値から更に中央値を特定する。
４．　データを次の３グループに分類する。前ステップで得た中央値（中央値の中央値）よりも値が小さい、等しい、大きい。
５．　前ステップの３グループから、グループ内の要素数から k 番目の要素を含むグループを特定し、そのグループに対しこのルーチンを再帰的にコールする。
k 番目の要素が前ステップの中央値と等しくない（小さい、または大きい）グループに属さない場合は、中央値と等しいグループに属するので、単に中央値を返す。

このアルゴリズムを VC++ で書くと次のようになります。

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

#define N 29
#define Q 5     // Assumes fixed value of five; Modifiy 'Q' functions
                    //  if value is other than five

int S[N];

#define swap(A,B) {int t; t = A; A = B; B = t;}

int SortLessThanQ(int S[], int num, int k)
{
    for (int i=0; i<num; i++) {
        for (int j=i+1; j<num; j++) {
            if (S[i] > S[j]) {
                swap(S[i], S[j]);
            }
        }
    }
    return S[k-1];
}

void CountAndMark(int S[], int Marks[], int num, int median, int leg[3])
{
    for (int i = 0; i < num; i++) {
        if (S[i] == median) {
```

```
            Marks[i] = 1; leg[1]++;
        } else if (S[i] < median) {
            Marks[i] = 0; leg[0]++;
        } else {
            Marks[i] = 2; leg[2]++;
        }
    }
}

void ArrayPack(int S[], int sPack[], int num, int Marks[], int scanSym)
{
    int j=0;
    for (int i = 0; i < num; i++)
        if (Marks[i] == scanSym) sPack[j++] = S[i];
}

int SequentialSelect(int *S, int num, int k)
{
    if (num <= Q) return SortLessThanQ(S, num, k);

    int cNum = num/Q + 1;
    int *Medians = new int[cNum];
    int i = 0;
    for (int j = 0; j < num/Q; j++) {
        Medians[j] = SortLessThanQ(&S[i], Q, (Q+1)/2);  // find medians of subsequences
        i += Q;
    }
    int lastNum = num - (Q * (num / Q));
    if (lastNum) {
        int lastQ = Q * (num / Q);
        Medians[cNum-1] = SortLessThanQ(&S[lastQ], lastNum, (lastNum+1)/2);
    } else
        cNum--;

    int M = SequentialSelect(Medians, cNum, (cNum+1)/2);
    delete Medians;

    int leg[3] = {0,0,0};
    int *markS = new int[num];
    CountAndMark(S, markS, num, M, leg);

    if (leg[0] >= k) {
        int *sPack = new int[leg[0]];
```

```
        ArrayPack(S, sPack, num, markS, 0);
        delete markS;
        M = SequentialSelect(sPack, leg[0], k);
        delete sPack;
        return M;
    } else if ((leg[0] + leg[1]) >= k) {
        delete markS;
        return M;
    } else {
        int *sPack = new int[leg[2]];
        ArrayPack(S, sPack, num, markS, 2);
        delete markS;
        M = SequentialSelect(sPack, leg[2], k-(leg[0]+leg[1]));
        delete sPack;
        return M;
    }
}

void init_data()
{
    int i, x, y;
    for (i = 0; i < N; i++){
        //S[i] = rand() % N;
        S[i] = i+1;
    }
    for (i = 0; i < N; i++){
        x = rand() % N;
        y = S[x]; S[x] = S[i]; S[i] = y;
    }
}

int main(int argc, char* argv[])
{

    init_data();

    for (int j = 0; j < N; j++){
        printf("%3d ",S[j]);
    }
    printf("\n\n");

    for (int r = 1; r <= N; r++) {
        int i = SequentialSelect(S, N, r);
```

```
        printf("r = %d i = %d\n", r, i);
        // init_data();
    }
    for (int j = 0; j < N; j++){
        printf("%3d ",S[j]);
    }
    printf("\n\n");

    return 0;
}
```

このように、アルゴリズムは１つではなく色々な方法があり、問題に応じてより良いアルゴリズムを探す必要があります。私は、今は手に入りませんが「Algorithms + Data Structures = Programs」Niklaus Wirth Prentice-Hall 1976 でアルゴリズムとデータ構造の基本とプログラミングの方法を学びました。自分にとってよい本に出会えるかどうかは運です。

例題：５０００未満の素数をすべて求めるプログラムを作れ。ここではエラトステネスの篩という有名なギリシャ時代から知られているアルゴリズムを使った例を示します。例えば、C で作ると次のようになります。

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{
    int i, k;
    int *p;

    if ((p = (int *)malloc(5001*sizeof(int))) == NULL) {
        printf("malloc error!\n");
        exit(1);
    }
    for (i=1; i<=5000; i++)
        p[i] = 1;
    p[1] = 0;
    for (i=4; i<=5000; i += 2)
        p[i] = 0;
    for (i=3; i <= sqrt((double)5000); i += 2) {
        if (!p[i])
            continue;
        for (k=2*i; k<=5000; k += i)
            p[k] = 0;
    }
    for (i=1; i<=5000; i++)
```

```
        if (p[i])
            printf("%4d ", i);
    free(p);
    getchar();
    return 0;
}
```

ここでは、malloc() による動的な配列を使っています。C++ で次のようにすることも出来ます。

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

using namespace std;

int main(int argc, char* argv[])
{
    int limit = 5000;
    bool p[5000];

    for (int i=2; i<limit; i++)
        p[i] = true;
    p[0] = false;
    p[1] = false;
    int ind = 2;
    while (ind < limit) {
        while (ind < limit && p[ind] == false)
            ind++;
        for (int k=2*ind; k<limit; k += ind)
            p[k] = false;
        ind++;
    }
    int n = 0;
    for (int i=2; i<limit; i++)
        if (p[i] == true) {
            cout << i << '\t';
            n++;
            if (n % 10 == 0) cout << endl;
        }
    getchar();
    return 0;
}
```

更に、vector を使ってプログラミングしてみます。

```
#include <iostream>
```

```
#include <vector>
#include <math.h>

using namespace std;

int main(int argc, char* argv[])
{
    int limit = 5000;
    vector<bool> p(5000);

    for(int i=0; i<limit; i++) p[i] = true;
    p[0] = p[1] = false;
    int ind = 2;
    while (ind <= sqrt((double)limit)) {
        while (ind < limit && p[ind] == false)
            ind++;
        for (int k=2*ind; k<limit; k += ind)
            p[k] = false;
        ind++;
    }
    int n = 0;
    for (int i=2; i<limit; i++)
        if (p[i] == true) {
            cout << i << '\t';
            n++;
            if (n % 10 == 0) cout << endl;
        }
    getchar();
    return 0;
}
```

と打ち込む。

```
#include <iostream>
#include <vector>
#include <math.h>

using namespace std;

int main(int argc, char* argv[])
{
    int limit = 5000;
    vector<bool> p(5000);
```

```
    for(int i=0; i<limit; i++) p[i] = true;
    p[0] = p[1] = false;
    int ind = 2;
    while (ind <= sqrt((double)limit)) {
        while (ind < limit && p[ind] == false)
            ind++;
        for (int k=2*ind; k<limit; k += ind)
            p[k] = false;
        ind++;
    }
    int n = 0;
    for (int i=2; i<limit; i++)
        if (p[i] == true) {
            cout << i << '\t';
            n++;
            if (n % 10 == 0) cout << endl;
        }
    getchar();
    return 0;
}
```

の

```
#include <iostream>
#include <vector>
#include <math.h>

using namespace std;
```

は今まで通りで、

```
int main(int argc, char* argv[])
{
    int limit = 5000;
    vector<bool> p(5000);

    for(int i=0; i<limit; i++) p[i] = true;
    p[0] = p[1] = false;
    int ind = 2;
    while (ind <= sqrt((double)limit)) {
        while (ind < limit && p[ind] == false)
            ind++;
        for (int k=2*ind; k<limit; k += ind)
            p[k] = false;
        ind++;
```

```
    }
    int n = 0;
    for (int i=2; i<limit; i++)
        if (p[i] == true) {
            cout << i << '\t';
            n++;
            if (n % 10 == 0) cout << endl;
        }
    getchar();
    return 0;
}
```

が main() 関数です。

```
    int limit = 5000;
    vector<bool> p(5000);

    for(int i=0; i<limit; i++) p[i] = true;
```

で、サイズ 5000 の bool 型のベクトル p を取り、初期化しています。BASIC のように

```
    int limit = 5000;
    vector<bool> p(limit);

    for(int i=0; i<limit; i++) p[i] = true;
```

でも良いです。0 と 1 は素数ではないので

```
    p[0] = p[1] = false;
```

とし、最初の素数候補を ind = 2 として、

```
    int ind = 2;
    while (ind <= sqrt((double)limit)) {
        while (ind < limit && p[ind] == false)
            ind++;
        for (int k=2*ind; k<limit; k += ind)
            p[k] = false;
        ind++;
    }
```

で、 ind が limit の平方根以下の間は、ind 以上で p[ind] == true となる最初の ind （この ind は素数）を探し、i = 2*ind, 3*ind, ... は素数でないので、p[i] = false とセットし、ind を 1 つ増やすを繰り返しています。これが終了すると p[i] == true なる i はすべて素数であるというのが「エラトステネスの篩」です。最後に

```
    int n = 0;
```

```
    for (int i=2; i<limit; i++)
        if (p[i] == true) {
            cout << i << '\t';
            n++;
            if (n % 10 == 0) cout << endl;
        }
```

で、結果を表示しています。

次は二次正方行列を入力し、その行列式を計算するプログラムである。

```
#include <stdio.h>
double det(double a[][3])
{
    return a[1][1]*a[2][2]-a[2][1]*a[1][2];
}
int main()
{
    double a[3][3];
    int i;

    for (i=1; i<=2; i++)
        for (j=1; j<=2; j++) {
            printf("a[%d][%d]=", i, j); scanf("%lf", &a[i][j]);
        }
    printf("行列式 = %f\n", det(a));
    getchar();
    getchar();
    return 0;
}
```

二次元の配列は `double a[3][3]` のように宣言する。この場合、

$$a[0][0], a[0][1], a[0][2], a[1][0], a[1][1], a[1][2], a[2][0], a[2][1], a[2][2]$$

の $3 \times 3 = 9$ 個にアクセスできるようになる。メモリが無駄になるので上のプログラムは次のように書く方がよい。引数として関数に渡すときは最初の大きさだけが省略できる。すなわち、関数 det の定義において、double det(double a[3][3]) と宣言してもよいが、最初の 3 だけは省略できる。

```
#include <stdio.h>
double det(double a[][2])
{
    return a[0][0]*a[1][1]-a[1][0]*a[0][1];
}
int main()
{
```

```
    double a[2][2];
    int i;

    for (i=0; i<2; i++)
        for (j=0; j<2; j++) {
            printf("a[%d][%d]=", i+1, j+1); scanf("%lf", &a[i][j]);
        }
    printf("行列式 = %f\n", det(a));
    getchar();
    getchar();
    return 0;
}
```

３次の正方行列の行列式の計算も行列式の定義をそのまま計算すればいいです。演習問題にしていますのでやってみてください。４次以上の正方行列の行列式の計算方法は大学の線形代数で学ぶので、その筆算のためのアルゴリズムをプログラミングすれば良いですが、複雑でちょっと厄介です。インターネットで調べるといろいろな人がいろいろなプログラミング言語でプログラミングしていますが、間違っている場合がありますので、インターネットで見つけたプログラムを使う場合は皆さんは教育学部で数学を学んでいる学生さんなので線形代数で行列式を計算する筆算のためのアルゴリズムを学ぶはずですから、学んだ通りのプログラムになっているか自分でしっかり正しいかどうか確かめてから使ってください。インターネットで検索して一番最初に出てくる情報がいつも正しいわけではありません。インターネットは誰でも情報を発信できます。悪意がなくてもも違っていることが結構あります。注意してください。次のようにします。

行列 A の i 行から、j 行の定数倍を引いても、行列式の値は変わらない。

という性質を使って行列 A を対角行列に変換する。行列式の定義を見ると、対角行列の行列式は、対角成分の積と等しい。

これをプログラミングすれば良いです。例えば、次のようになります。

```
#include "StdAfx.h"
#include <iostream>
#include <stdio.h>
#include <math.h>

using namespace std;

int main() {
    int n;
    cout << "n="; cin >> n;

    double **A;
    A = new double*[n];
    for (int i=0; i<n; i++) [
        A[i] = new double[n];
    }
```

```
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) {
        cout << "A[" << i << "][" << j << "]="; cin >> A[i][j];
    }
}
for (int i=0; i<n; i++) {
    int k;
    if (A[i][i] == 0) {
        for (k=0; k<n; k++) {
            if (A[k][i] != 0) {
                break;
            }
        }
        if (k == n) {
            cout << " 行列式 = " << 0 << endl;
            for (int i=0; i<n; i++)
                delete[] A[i];
            delete[] A;
            getchar();
            getchar();
            return 0;
        }
        for (int j=0; j<n; j++) {
            A[i][j] += A[k][j];
        }
    }
    for (int j=i+1; j<n; j++) {
        double C = A[i][j] / A[i][i];
        for (int k=0; k<n; k++) {
            A[k][j] -= C * A[k][i];
        }
    }
}
double det = 1;
for (int k=0; k<n; k++)
    det *= A[k][k];
cout << " 行列式 = " << det << endl;
for (int i=0; i<n; i++)
    delete[] A[i];
delete[] A;
getchar();
getchar();
return 0;
```

```
}
```

配列を使えば、大きな桁数の計算が出来るようになります。ここでは「C -言語とプログラミング-」米田信夫編　産業図書にある π の計算のプログラムを研究する事にします。

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

#define RADIX 10
#define W 1100

using namespace std;

void init(int a[], int n)
{
    int i;

    a[0] = n;
    for (i=1; i<=W; i++)
        a[i] = 0;
}

void add(int a[], int b[], int c[])
{
    int t, x, i;

    t = 0;
    for (i=W; i>=0; i--) {
        x = b[i] + c[i] + t;
        a[i] = x % RADIX;
        t = x / RADIX;
    }
    if (t != 0) {
        cout << "overflow\n";
    }
}

void sub(int a[], int b[], int c[])
{
    int t, x, i;

    t = 1;
    for (i=W; i>=0; i--) {
        x = b[i] + (RADIX - 1 - c[i]) + t;
```

```
        a[i] = x % RADIX;
        t = x / RADIX;
    }
    if (t != 1)
        cout << "overflow\n";
}

int top(int a[], int p)
{
    while (p <= W && a[p] == 0)
        p++;
    return p;
}

void div(int a[], int b[], int n)
{
    int t, x, i;

    t = 0;
    for (i=0; i<=W; i++) {
        x = t * RADIX + b[i];
        a[i] = x / n;
        t = x % n;
    }
}

void marctan(int a[], int n, int d)
{
    int e[W+1], f[W+1];
    int p, i;

    init(a, 0);
    init(e, n);
    div(e, e, d);
    p = top(e, 0);
    add(a, a, e);
    i = 3;
    while (p <=W) {
        div(e, e, d);
        div(e, e, d);
        div(f, e, i);
        if (i % 4 == 1)
            add(a, a, f);
```

```
        else
            sub(a, a, f);
        p = top(e, p);
        i += 2;
    }
}

int main()
{
    int a[W+1], b[W+1], pi[W+1];
    int i;

    marctan(a, 16, 5);
    marctan(b, 4, 239);
    sub(pi, a, b);
    printf("%d.", pi[0]);
    for (i=1; i<=1000; i++) {
        if (i%50==1 && i != 1)
            printf("  ");
        printf("%d", pi[i]);
        if (i % 5 == 0)
            putchar(' ');
        if (i % 50 == 0)
            putchar('\n');
    }
    getchar();
    return 0;
}
```

π の小数点以下１０００桁は

```
3.
14159 26535 89793 23846 26433 83279 50288 41971 69399 37510
58209 74944 59230 78164 06286 20899 86280 34825 34211 70679
82148 08651 32823 06647 09384 46095 50582 23172 53594 08128
48111 74502 84102 70193 85211 05559 64462 29489 54930 38196
44288 10975 66593 34461 28475 64823 37867 83165 27120 19091
45648 56692 34603 48610 45432 66482 13393 60726 02491 41273
72458 70066 06315 58817 48815 20920 96282 92540 91715 36436
78925 90360 01133 05305 48820 46652 13841 46951 94151 16094
33057 27036 57595 91953 09218 61173 81932 61179 31051 18548
07446 23799 62749 56735 18857 52724 89122 79381 83011 94912
98336 73362 44065 66430 86021 39494 63952 24737 19070 21798
60943 70277 05392 17176 29317 67523 84674 81846 76694 05132
```

```
00056 81271 45263 56082 77857 71342 75778 96091 73637 17872
14684 40901 22495 34301 46549 58537 10507 92279 68925 89235
42019 95611 21290 21960 86403 44181 59813 62977 47713 09960
51870 72113 49999 99837 29780 49951 05973 17328 16096 31859
50244 59455 34690 83026 42522 30825 33446 85035 26193 11881
71010 00313 78387 52886 58753 32083 81420 61717 76691 47303
59825 34904 28755 46873 11595 62863 88235 37875 93751 95778
18577 80532 17122 68066 13001 92787 66111 95909 21642 01989
```

となります。

このπの計算にはマチンの公式

$$\frac{\pi}{4} = 4 \cdot \arctan\frac{1}{5} - \arctan\frac{1}{239}$$

を使います。$\arctan x$ は

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} + \cdots$$

で計算できます。従って

$$\pi = 16 \cdot \arctan\frac{1}{5} - 4 \cdot \arctan\frac{1}{239}$$

なので

$$\pi = \frac{16}{5} - \frac{\frac{16}{5}\cdot(\frac{1}{5})^2}{3} + \frac{\frac{16}{5}\cdot(\frac{1}{5})^2\cdot(\frac{1}{5})^2}{5} - \frac{\frac{16}{5}\cdot(\frac{1}{5})^2\cdot(\frac{1}{5})^2\cdot(\frac{1}{5})^2}{7}$$
$$+\frac{\frac{16}{5}\cdot(\frac{1}{5})^2\cdot(\frac{1}{5})^2\cdot(\frac{1}{5})^2\cdot(\frac{1}{5})^2}{9} - \frac{\frac{16}{5}\cdot(\frac{1}{5})^2\cdot(\frac{1}{5})^2\cdot(\frac{1}{5})^2\cdot(\frac{1}{5})^2\cdot(\frac{1}{5})^2}{11} + \cdots$$
$$-\frac{4}{239} + \frac{\frac{4}{239}\cdot(\frac{1}{239})^2}{3} - \frac{\frac{4}{239}\cdot(\frac{1}{239})^2\cdot(\frac{1}{239})^2}{5} + \frac{\frac{4}{239}\cdot(\frac{1}{239})^2\cdot(\frac{1}{239})^2\cdot(\frac{1}{239})^2}{7}$$
$$-\frac{\frac{4}{239}\cdot(\frac{1}{239})^2\cdot(\frac{1}{239})^2\cdot(\frac{1}{239})^2\cdot(\frac{1}{239})^2}{9} + \frac{\frac{4}{239}\cdot(\frac{1}{239})^2\cdot(\frac{1}{239})^2\cdot(\frac{1}{239})^2\cdot(\frac{1}{239})^2\cdot(\frac{1}{239})^2}{11} - \cdots$$

を計算すれば良いことになります。

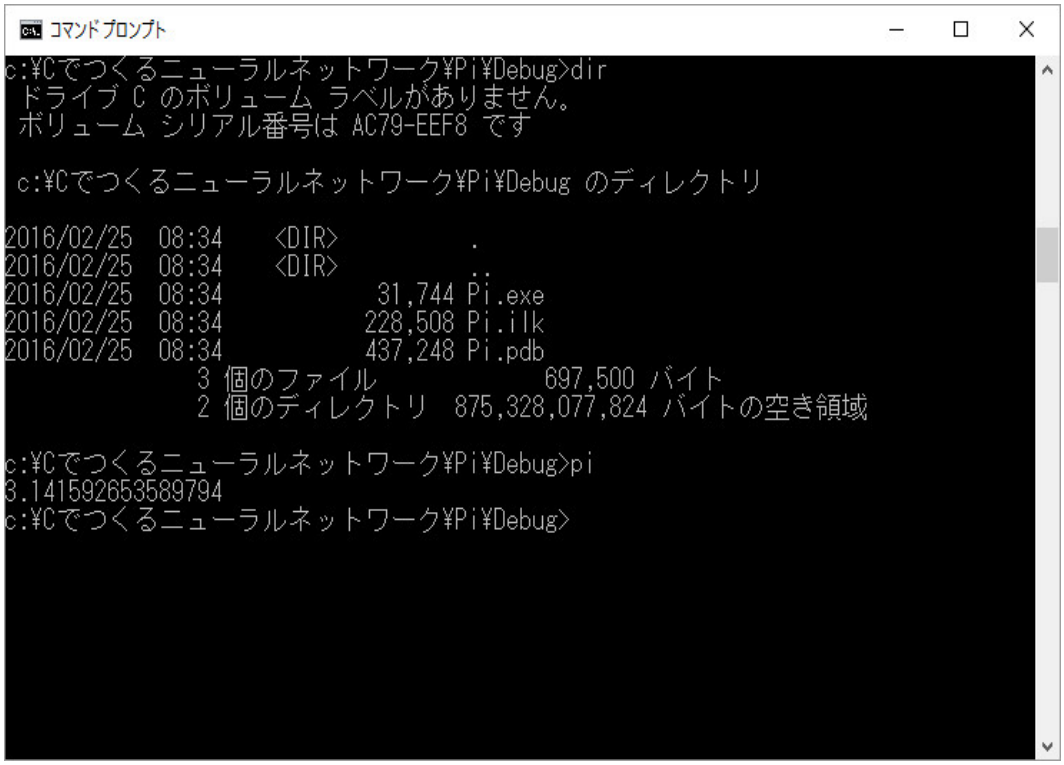
この式通り計算するプログラムは

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



double marctan(int n, int d) {
    double a = 0.0;
    double e = double(n);
    double f = 0.0;
    e = e / d;
    a = a + e;
    int i = 3;
    while (e > 0.000000000000001) {
        e = e / d;
```

```
        e = e / d;
        f = e / i;
        if (i % 4 == 1) {
            a = a + f;
        }
        else {
            a = a - f;
        }
        i = i + 2;
    }
    return a;
}

int main() {
    double a = marctan(16, 5);
    double b = marctan(4, 239);
    double pi = a - b;
    printf("%.15f", pi);
}
```

です。

実行すると



です。

この計算を double ではなく、大きな桁数の固定小数点数で計算するためには、多きな桁数の数の足し算と引き算と割り算があれば良い事が分かります。更に、求める円周率の桁数がそれほど大きくなければ（$\arctan x$ を求めるための分子が int の範囲にあるならば)、割り算は多きな桁数の数を単独の整数で割る割り算で良い事が見て取れます。

足し算は

```
void add(int a[], int b[], int c[])
{
    int t, x, i;

    t = 0;
    for (i=W; i>=0; i--) {
        x = b[i] + c[i] + t;
        a[i] = x % RADIX;
        t = x / RADIX;
    }
    if (t != 0) {
        cout << "overflow\n";
    }
}
```

で、計算できます。

```
#define RADIX 10
#define W 1100
```

は、１０進数で、桁数は 1100 であることを示しています。これは１００進数でも 1000 進数でも良いですが、我々になじみ深い１０進数にしてあります。アルゴリズムは小学校で習う１０進数の足し算と同じものです。

このプログラムで注目すべきは

```
void sub(int a[], int b[], int c[])
{
    int t, x, i;

    t = 1;
    for (i=W; i>=0; i--) {
        x = b[i] + (RADIX - 1 - c[i]) + t;
        a[i] = x % RADIX;
        t = x / RADIX;
    }
    if (t != 1)
        cout << "overflow\n";
}
```

の部分です。

このアルゴリズムは小学校で習う１０進数の引き算とは異なります。小学校で習う１０進数のくり下がりの引き算は「減加法」とか「減々法」のどちらを教えるべきかの議論がありますが、引き算には、このような計算方法（アルゴリズム）もあります。機械語の講義で、負の数を表すのに２の補数というのを学びました。このアルゴリズムも同じことをしています。２進数の場合は２の補数 (1 と 0 を入れ替え 1 を足す。即ち、各数字を 1 から引き 1 を足す) でしたが、今は１０進数ですから１０の補数（各数字を 9 から引き 1 を足す）を使います。例えば、$9536 - 5814$ を計算するには、5814 を１０の補数 $4185 + 1$ にし、$9536 + 4186 = 13722$ を計算し、一番上の 1 を除いた 3722 が答えになります。$9102 - 3234$ の計算は 3234 の１０の補数 $6765 + 1$ を 9102 に加え $9102 + 6766 = 15868$ の一番上の 1 を除いた 5868 が答えになります。これが正しいのは、

$$9102 - 3234 = 9102 - 3234 + 10000 - 10000 = 9102 + 9999 + 1 - 3234 - 10000$$
$$= 9102 + (9999 - 3234) + 1 - 10000 = 9102 + (6765 + 1) - 10000 = 15868 - 10000 = 5868$$

から、理解できます。このアルゴリズムでは、くり下がりがあるかどうかの判断（ if 文）が不要になります。

割り算は

```
void div(int a[], int b[], int n)
{
    int t, x, i;

    t = 0;
    for (i=0; i<=W; i++) {
        x = t * RADIX + b[i];
        a[i] = x / n;
        t = x % n;
    }
}
```

です。これは小学校で習う割り算と本質的に同じです。桁数の多い数を桁数の多い数で割る割り算は試行錯誤が必要で難しいです。割り算の効率の良いアルゴリズムは Knuth の「The Art of Computer Programming」 にアルゴリズムが載っています。

```
int top(int a[], int p)
{
    while (p <= W && a[p] == 0)
        p++;
    return p;
}
```

で、配列 a の 0 でない最初のインデックスを求めています。これは、a が小さな数かを判定するのに使います。

```
void init(int a[], int n)
{
    int i;
```

```
    a[0] = n;
    for (i=1; i<=W; i++)
        a[i] = 0;
}
```

で、配列 a に、$a = \{n, 0, \cdots, 0\}$ をセットしています。
　これらの関数を使って

```
void marctan(int a[], int n, int d)
{
    int e[W+1], f[W+1];
    int p, i;

    init(a, 0);
    init(e, n);
    div(e, e, d);
    p = top(e, 0);
    add(a, a, e);
    i = 3;
    while (p <=W) {
        div(e, e, d);
        div(e, e, d);
        div(f, e, i);
        if (i % 4 == 1)
            add(a, a, f);
        else
            sub(a, a, f);
        p = top(e, p);
        i += 2;
    }
}

int main()
{
    int a[W+1], b[W+1], pi[W+1];
    int i;

    marctan(a, 16, 5);
    marctan(b, 4, 239);
    sub(pi, a, b);
    printf("%d.", pi[0]);
    for (i=1; i<=1000; i++) {
        if (i%50==1 && i != 1)
```

```
            printf("  ");
        printf("%d", pi[i]);
        if (i % 5 == 0)
            putchar(' ');
        if (i % 50 == 0)
            putchar('\n');
    }
    getchar();
    return 0;
}
```

で、円周率を計算しています。

π の計算の歴史 アルキメデス（紀元前 283-212 年）は $n = 6, 12, 24, 48, 96$ の正 n 角形の長さを計算し、半角の公式

$$\sin(\frac{x}{2}) = \pm\sqrt{\frac{1-\cos x}{2}}, \cos(\frac{x}{2}) = \pm\sqrt{\frac{1+\cos x}{2}}$$

を繰り返し使うことにより

$$3\frac{10}{71} < \pi < 3\frac{1}{7}$$

という評価を得ました。この値を改良しようとする中世のあらゆる試みは実を結びませんでしたが、ついに、アドリアン・ファン・ルーメンが（1580 年に）、アルキメデスの方法を使い、何年も計算した末に小数点以下 20 桁までの値を得ることに成功しました。ルドルフ・ファン・ケーレンは（1596, 1616 年に）35 桁まで計算しました。この精度に達するために、ルドルフは $n = 6\dot{2}^{60}$ まで計算を続けなければなりませんでした。ドイツでは π のことをルドルフの数と言います。

日本では、江戸時代の１７１２年に関孝和が円周率を１１桁、１７２２年に建部賢弘が４１桁、さらに１７３９年刊の「方円算経」で松永良弼が５２桁計算しました。建部賢弘の円周率公式は

$$\frac{\pi^2}{9} = 1 + \frac{1^2}{3\cdot 4} + \frac{1^2\cdot 2^2}{3\cdot 4\cdot 5\cdot 6} + \frac{1^2\cdot 2^2\cdot 3^2}{3\cdot 4\cdot 5\cdot 6\cdot 7\cdot 8} + \cdots$$

で、松永良弼の円周率公式は

$$\frac{\pi}{3} = 1 + \frac{1^2}{4\cdot 6} + \frac{1^2\cdot 3^2}{4\cdot 6\cdot 8\cdot 10} + \frac{1^2\cdot 3^2\cdot 5^2}{4\cdot 6\cdot 8\cdot 10\cdot 12\cdot 14} + \cdots$$

です。

ライプニッツが公式

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} + \cdots$$

を発見します。この式で $x = 1$ と置けば、有名なライプニッツの公式

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \cdots$$

（1682 年）が得られます。この式は美しさとは裏腹に、あまり役に立ちません。なにしろ、50 桁欲しければ 10^{50} 項の計算を「永遠に続ける努力で」（オイラー 1737）、行わねばなりません。ずっと有効なのは $\tan(\pi/6) = 1/\sqrt{3}$ を使うことです。すると

$$\pi = 2\sqrt{3}(1 - \frac{1}{3\cdot 3} + \frac{1}{5\cdot 3^2} - \frac{1}{7\cdot 3^3} + \cdots)$$

という公式が得られます。この式を用いて、Th.F. ド・ラニーは、「信じられないほどの仕事量を示している」210 項を加えることによって、127 桁（113 桁目に間違いがあった）を 1719 年に計算しました。

$$\tan(x+y) = \frac{\tan x + \tan y}{1 - \tan x \tan y}$$

の式に $u = \tan x, v = \tan y$ を代入すれば、$|\arctan u + \arctan v| < \pi/2$ のとき、

$$\arctan u + \arctan v = \arctan(\frac{u+v}{1-uv})$$

が得られます。ここで、$u = 1/2, v = 1/3$ を代入すれば、オイラーの公式 (1737 年)

$$\frac{\pi}{4} = \arctan\frac{1}{2} + \arctan\frac{1}{3}$$

が得られます。有名なジョン・マチンの公式は次のようにして求めます。$u = v = 1/5$ と置けば

$$2 \cdot \arctan\frac{1}{5} = \arctan(\frac{2/5}{1-1/25}) = \arctan\frac{5}{12}$$

が得られます。$u = v = 5/12$ と置けば

$$2 \cdot \arctan\frac{5}{12} = \arctan(\frac{5/6}{1-25/144}) = \arctan\frac{120}{119}$$

となります。そこで、$u = 120/119$ と置いて、v を

$$\frac{u+v}{1-uv} = 1 \text{ となるように、したがって } v = \frac{1-u}{1+u} = -\frac{1}{239}$$

とおきます。これらの式をまとめると、

$$\frac{\pi}{4} = 4 \cdot \arctan\frac{1}{5} - \arctan\frac{1}{239}$$

となります。この公式を使ってマチンは 100 桁の値を求めたことを W. ジョーンズ（1706 年）が細かい点は述べずに紹介しています。

最近のコンピュータによる円周率の計算では

$$\pi = 48 \cdot \arctan\frac{1}{49} - 128 \cdot \arctan\frac{1}{57} - 20 \cdot \arctan\frac{1}{239} + 48 \cdot \arctan\frac{1}{110443}$$

や

$$\pi = 176 \cdot \arctan\frac{1}{57} + 28 \cdot \arctan\frac{1}{239} - 48 \cdot \arctan\frac{1}{682} + 96 \cdot \arctan\frac{1}{12943}$$

という式が使われたそうです。

最後に N 人の女王のプログラムを再帰を使った C のプログラムを載せます。

```
#include <stdio.h>

int mm[9], jj[9];
int kk[16], ll[16];

void queens(int m, int n)
{
```

```
    int c, i;

for (c=1; c<=m; c++) {
        if (!(jj[c]==0 && kk[n+1-c+m]==0 && ll[n+1+c-1]==0))
            continue;
            mm[n+1] = c;
            jj[c] = kk[n+1-c+m] = ll[n+c] = 1;
            if (n+1 == m) {
                for (i=1; i<=m; i++)
                    printf("%d ", mm[i]);
                printf("\n");
            }
        else
            queens(m, n+1);
        jj[c] = kk[n+1-c+m] = ll[n+c] = 0;
    }
}

int main()
{
    int n, i;

printf("n=");
    scanf("%d", &n);
    for (i=0; i<16; i++)
        kk[i] = ll[i] = 0;
    for (i=0; i<9; i++)
        jj[i] = 0;
    queens(n, 0);
getchar(); getchar();
return 0;
}
```

このプログラムでは

```
#include <stdio.h>

int mm[9], jj[9];
int kk[16], ll[16];
```

と配列の宣言を関数の外で宣言しています。このようにするとすべての関数でこれら配列が使えます。

ちなみに N 人の女王の FORTRAN のプログラムの例は次のようなものです。

```
      DIMENSION MM(8),JJ(8),KK(15),LL(15)
```

```
      DO 10 IX=1,8
   10    JJ(IX)=0
      DO 20 IX=1,15
         KK(IX)=0
   20    LL(IX)=0
      I=1
   30 N=0
   40 N=N+1
      IF(N.GT.8) GOTO 60
      IF(JJ(N).EQ.1) GOTO 40
      IF(KK(I-N+8).EQ.1) GOTO 40
      IF(LL(I+N-1).EQ.1) GOTO 40
      MM(I)=N
      JJ(N)=1
      KK(I-N+8)=1
      LL(I+N-1)=1
      I=I+1
      IF(I.LE.8) GOTO 30
      WRITE(6,50)(MM(IY),IY=1,8)
   50 FORMAT(5H ANS=,8I5)
   60 I=I-1
      IF(I.LT.1) STOP
      N=MM(I)
      JJ(N)=0
      KK(I-N+8)=0
      LL(I+N-1)=0
      GOTO 40
      END
```

FORTRAN では再帰が使えなかったので、理解するのが難しいプログラムになります。

演習問題

1. 三次元のベクトルを配列に入力し、その長さ（ノルム）を計算するプログラムを書け。
2. 三次元のベクトルを 2 個、配列に入力し、その内積を計算するプログラムを書け。
3. 三次正方行列を配列に入力し、その行列式を計算するプログラムを書け。
4. 二次正方行列を二個入力し、その積を計算するプログラムを書け。

0.3 方程式の根の近似解法

昔の数学 B の教科書には正の数 a の平方根 $\sqrt{a}$ を求める次のようなアルゴリズムが載っている。

まず、縦の長さが 1、横の長さが a の長方形からはじめ面積 a を変えないで、ある規則にしたがって縦と横の長さだけ変え、正方形に近づけていく。正方形が得られたとき、その辺の長さを x

とすれば、 $x^2 = a$ から平方根 $\sqrt{a}$ を求めることができる。長方形を正方形に近づける手順において、k 回目の縦、横の長さをそれぞれ y_k, x_k とする。縦と横の長さの平均

$$x_{k+1} = \frac{x_k + y_k}{2}$$

によって k+1 回目の横の長さを定める。$x_{k+1} \cdot y_{k+1} = a$ から縦の長さは

$$y_{k+1} = \frac{a}{x_{k+1}}$$

である。

$x_0 = a, y_0 = 1$ からはじめ、上の計算を続け、数列 $\{x_k\}, \{y_k\}$ を求め k を大きくしていくと、それぞれの数列が平方根 $\sqrt{a}$ に限りなく近づく。

この算法によるプログラムは、次のようになる。

```
PRINT "長方形から正方形による平方根の値"
INPUT "縦=1, 横の長さ A を入力 A=";A
PRINT "      X               Y"
X=A : Y = 1
FOR I=1 TO 5
  X=(X+Y)/2
  Y=A/X
  PRINT USING "0.0000000000000000" X, USING "0.0000000000000000" Y
NEXT I
END
```

実行すると

```
長方形から正方形による平方根の値
縦=1, 横の長さ A を入力 A=3
      X               Y
2.0000000000000000      1.5000000000000000
1.7500000000000000      1.7142857142857142
1.7321428571428572      1.7319587628865978
1.7320508100147274      1.7320508051230272
1.7320508075688772      1.7320508075688774
```

となる。わずか 5 回の繰り返しで、小数大 1 5 位まで一致し、良い近似値が得られている。

なお、この算法では、平方根を求めるわけであるから、x_k, y_k の両方を求める必要はない。$y_k = \frac{a}{x_k}$ を $x_{k+1} = \frac{x_k + y_k}{2}$ に代入して

$$x_{k+1} = \frac{1}{2}(x_k + \frac{a}{x_k})$$

が得られる。これを使えばよい。これは数学 C にあるニュートン法の特別な場合である。

この算法による C++ のプログラムは、次のようになる。

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

using namespace std;

int main(int argc, char* argv[])
{
    double a, x, y;

    cout << "a="; cin >> a;
    getchar();
    x = a; y = 1.0;
    for (int i=0; i<5; i++) {
        x = (x + y) / 2;
        y = a / x;
        cout << x << "\t" << y << endl;
    }
    getchar();
    return 0;
}
```

一般的に方程式の近似解を求めるには二分法とニュートン法がよく使われる。これは昔の数学 C で解説されている。一般的に方程式の近似解を求めるには二分法とニュートン法がよく使われる。これは数学 C で解説されている。次は二分法により平方根を求める旺文社数学 C の BASIC のプログラムである。

```
100 REM 二分法
110 EPS = 0.00001
120 INPUT "S=";S
130 DEF FNY(X)=X^2-S
140 IF S<1 THEN P=0:Q=1 ELSE P=1:Q=S
150 FOR N=1 TO 20
160   R=(P+Q)/2:NI=N
170   IF FNY(R)*FNY(Q)<0 THEN P=R ELSE Q=R
180   IF ABS(Q-P)<=EPS THEN 200
190 NEXT N
200 PRINT NI;"回反復　近似値＝";R
210 END
```

実行すると

```
S=3
18.0 回反復　近似値＝ 1.7320480346679688
```

となる。

s の平方根を求めるには $f(x) = x^2 - s = 0$ の解を求めればよい。s の値により $f(p) < 0,\ f(q) > 0$ なる p, q を初期値として与える。

C++ で二分法により平方根を求めてみよう。s の平方根を求めるには $f(x) = x^2 - s = 0$ の解を求めればよい。s の値により $f(p) < 0,\ f(q) > 0$ なる p, q を初期値として与える。

```
#include <iostream>
#include <stdio.h>
#include <math.h>

using namespace std;

double fn(double x, double s)
{
    return x * x - s;
}

int main(int argc, char *argv[])
{
    double s, eps = 0.0000001;
    double p, q, r;

    cout << "s="; cin >> s; getchar();
    if (s < 1) {
        p = 0; q = 1;
    } else {
        p = 1; q = s;
    }
    int n;
    for (n=0; n<200; n++) {
        r = (p + q) / 2;
        if (fn(r, s) * fn(q, s) < 0)
            p = r;
        else
            q = r;
        if (fabs(q - p) < eps)
            break;
    }
    cout << n+1 << "回反復　近似値＝" << p << endl;
    cout << "p*p=" << p * p << endl;
    getchar();
    return 0;
}
```

次は $f(x) = x^2 - \cos x = 0$ の近似解を求める二分法のプログラムである。二分法ではグラフを

描くとか何らかの方法で $f(a) \times f(b) < 0$ となる初期値 a, b を見つけなければならない。

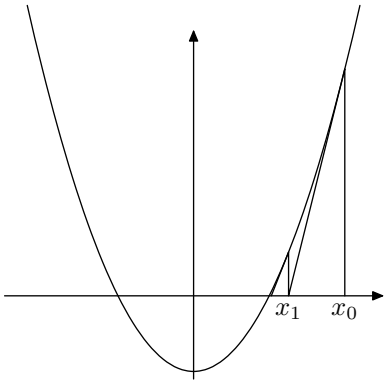
```
#include <iostream>
#include <stdio.h>
#include <math.h>

using namespace std;

double fn(double x)
{
    return x * x - cos(x);
}

int main(int argc, char* argv[])
{
    double a, b, m, eps = 0.00000000001;

    a = 0.5; b = 2.0; m = (a+b)/2;
    cout << "fn(" << a << ")=" << fn(a) << endl;
    cout << "fn(" << b << ")=" << fn(b) << endl;
    while (fabs(fn(a)-fn(b)) > eps) {
        m = (a+b)/2;
        if (fn(m) > 0) b = m; else a = m;
    }
    cout << "SOLUTION=" << m << endl;
    getchar();
    return 0;
}
```

次に Newton-Raphson 法で $f(x) = x^2 - \cos x$ の解を求める。

$f(x) = 0$ の解 x の近似解を x_k とし、Taylor 級数展開の x に関して一次までの項で $f(x)$ を近似すると、

$$f(x) = f(x_k) + f'(x_k)(x - x_k) = 0$$

を得る。これより $x_{k+1} = x_k - f(x_k)/f'(x_k)$ の逐次計算を行うと、ある条件のもとで真の解に収束する。



方程式によっては、ニュートン法では、求めたい解の近くに初期値を取っても、別の解の近似値が求まったり、近似解がまったく求められない場合もある。

```
#include <iostream>
#include <stdio.h>
#include <math.h>

using namespace std;

double func(double x)
{
    return x * x - cos(x);
}

double diff(double x)
{
    return 2 * x + sin(x);
}

int main(int argc, char* argv[])
{
    double x, dx;
    int i, max = 100;

    cout << "初期値？"; cin >> x; getchar();
    i = 0;
    do {
        dx = - func(x) / diff(x);
        x = x + dx;
    } while (i <= max && fabs(dx) > 0.0000001);
    cout << "近似解=" << x << endl;
    getchar();
    return 0;
```

```
}
```

同じことを Java でプログラミングすると次のようになる。

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Newton {
    static double f(double x) {
        return x * x - Math.cos(x);
    }
    static double df(double x) {
        return 2 * x + Math.sin(x);
    }
    public static void main(String[] args) {
        int i, max = 100;
        double x, dx;

        BufferedReader rd =
            new BufferedReader(new InputStreamReader(System.in));
        try {
            String line;
            System.out.print("初期値？");
            line = rd.readLine();
            x = Double.parseDouble(line);
        }
        catch(IOException e) {
            System.out.println("入力エラーが発生しました。");
            return; // 入力エラーなら終了する
        }
        catch(NumberFormatException e) {
            System.out.println("実数を入力して下さい。");
            return; // 入力ミスなら終了する
        }
        System.out.println("  i       x            f(x)         df(x)");
        i=0;
        do {
            dx = - f(x) / df(x);
            x += dx;
            System.out.println(i+","+x+","+f(x)+","+df(x));
            i++;
        } while((i <= max) && (Math.abs(dx) > 0.00000001));
        System.out.println("解="+x);
```

```
    }
}
```

実行すると

```
初期値？2
  i       x             f(x)          df(x)
0,1.1004523758499203,0.75780251717421,3.0923172164951502
1,0.8553926151203572,0.07577432278402518,2.465613729520581
2,0.8246601764269862,0.0012578634935394017,2.383637503039626
3,0.8241324688825565,3.730086470810079E-7,2.382223774388821
4,0.8241323123025361,3.26405569239796E-14,2.382223354880568
5,0.8241323123025225,2.220446049250313E-16,2.3822233548805314
解=0.8241323123025225
```

となる。

注　上の図は

```
prologues := 1;
defaultfont := "rptmr";
defaultscale := 1.5;
beginfig(1);
u=1cm;
vardef f(expr x)=x*x-cosd(x) enddef;
vardef g(expr x)=2*x+sind(x) enddef;
drawarrow (-2.5u,0)--(2.5u,0);
drawarrow (0,-1.1u)--(0,3.5u);
numeric a, b, c;
a := -2.2;
draw (a*u,f(a)*u)
for i=-2.2 step 0.2 until 2.3: ..(i*u, f(i)*u) endfor;
b := 2-f(2)/g(2);
c := b-f(b)/g(b);
draw (2u,0)--(2u,f(2)*u)--(b*u,0)--(b*u,f(b)*u)--(c*u,0);
label.bot(btex $x_0$ etex, (2u,0));
label.bot(btex $x_1$ etex, (b*u,0));
endfig;
end.
```

というプログラムで、 MetaPost を使って作画した。MetaPost は John Hobby が作った METAFONT 類似のソフトで、 PostScript（EPS）形式のファイルを出力します。 1995 年から AT&T ライセンス不要のフリーソフトになりました。METAPOST は METAFONT をお手本にして作られたプログラミング言語で、 METAFONT は白黒の図形しか作れないのに対して、METAPOST ならフルカラーの図形が作れるなどの細かい違いはありますが、共通している部分が多いです。MetaPost は LaTeX を導入すれば、多分自動的にインストールされている。METAFONT は Donald E.

Knuth が作ったフォント作成ツールで、 METAFONT を使って色々な gf 形式のフォントが作れます。 METAFONT を使い、色々の人の無料奉仕で、アラビア語やロシア語やギリシャ語や漢文など色々な言語が LaTeX で使えるようになっています。更に、LaTeX は数式や楽譜や化学式などが綺麗に書ける無料のワープロソフトで、数学の先生は全員使っています。このテキストも LaTeX で作っています。

発展問題：ルート 2 の計算の歴史

$\sqrt{2}$ のもっと上手な計算法　$\sqrt{a+b}$ において、b が a に比べて小さいものと考えると、$\sqrt{a+b} \approx \sqrt{a}$ となります。$\sqrt{a+b}$ と $\sqrt{a}$ との差である未知量 δ を考えて近似を良くしようとすると、

$$\sqrt{a+b} = \sqrt{a} + \delta$$

より

$$a+b = \left(\sqrt{a}+\delta\right)^2 = a + 2\sqrt{a}\delta + \delta^2$$

となります。δ が小さいから、δ^2 を無視すれば、$\delta = b/(2\sqrt{a})$ が得られ、したがって、

$$\sqrt{a+b} \approx \sqrt{a} + \frac{b}{2\sqrt{a}} (|b| \ll |a| \text{ のとき})$$

が得られます。さて、$\sqrt{2}$ の近似値として $v = 1.4$ から始めることにして、$a = v^2, b = 2-a = 2-v^2$ とおきます。すると、上の式より新しい近似値として

$$v + \frac{2-v^2}{2v} = \frac{1}{2}(\frac{2}{v} + v)$$

が得られ、この公式を次々と使うことが出来て、

1.4

1.414285

1.4142135642

1.4142135623730950499

1.41421356237309504880168872420969807 90

が得られます。まったく同じ計算を 60 進法で行うことができ、$1,25$ (即ち $1+\frac{25}{60}$) で始めると $1,24,51,10$ (即ち $1+\frac{24}{60}+\frac{51}{60^2}+\frac{10}{60^3}$) が得られます。この値が紀元前 1900 年のバビロニアの粘土板で見つかるのです。

1450 年頃アルカルサーディーが上の式を改良します。

$$\sqrt{a+b} = \sqrt{a} + \frac{b}{2\sqrt{a}} + \delta$$

を考えて、平方を計算すると、

$$a+b = a+b+\frac{b^2}{4a} + 2\sqrt{a}\delta + \frac{b\delta}{\sqrt{a}} + \delta^2$$

が得られます。最後の 2 項を無視すれば、

$$\sqrt{a+b} \approx \sqrt{a} + \frac{b}{2\sqrt{a}} - \frac{b^2}{8\sqrt{a^3}}$$

が得られます。$\sqrt{2}$ に対して今度は

$$v + \frac{2-v^2}{2v} - \frac{4-4v^2+v^4}{8v^3} = \frac{3v}{8} + \frac{3}{2v} - \frac{1}{2v^3}$$

が得られ、$v = 1.4$ に次々と使っていけば、速く収束する近似列

1.4

1.4142128

1.41421356237309504870

1.41421356237309504880168872420969807856967187537694807317643

が得られます。上の近似式は $\sqrt{a}$ で割って、$\frac{b}{a}$ を x で置き換えると極めて簡潔な式

$$\sqrt{1+x} \approx 1 + \frac{x}{2}, \sqrt{1+x} \approx 1 + \frac{x}{2} - \frac{x^2}{8}$$

になります。もっと精密な近似式

$$\sqrt{1+x} = 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \frac{1}{16}x^3 - \frac{5}{128}x^4 + \cdots$$

をニュートンが 1665 年に見つけます。

発展問題：複素関数のニュートン法

Newton-Raphson 法は複素関数に対してもまったく同様に使えます。

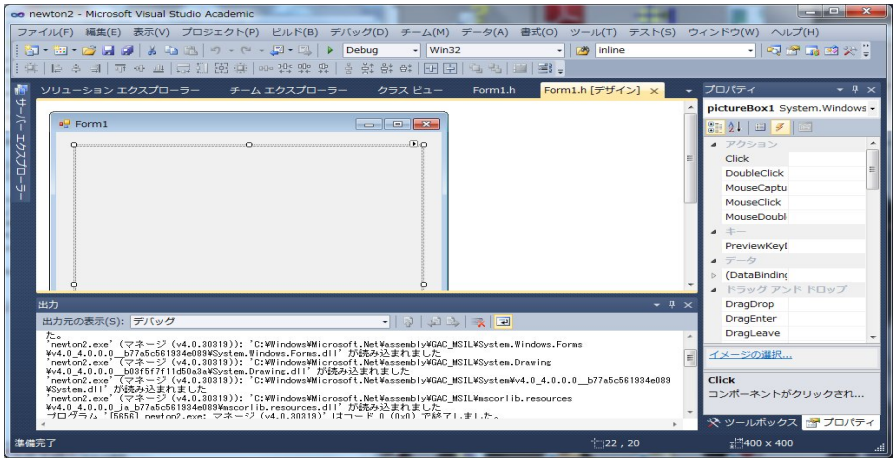
$f(z) = 0$ の解 z の近似解を z_k とし、Taylor 級数展開の z に関して一次までの項で $f(z)$ を近似すると、

$$f(z) = f(z_k) + f'(z_k)(z - z_k) = 0$$

を得る。これより $z_{k+1} = z_k - f(z_k)/f'(z_k)$ の逐次計算を行うと、ある条件のもとで真の解に収束する。

方程式によっては、ニュートン法では、求めたい解の近くに初期値を取っても、別の解の近似値が求まったり、近似解がまったく求められない場合もある。$f(z) = z^3 - 1$ の根は $z = 1, z = -\frac{1}{2} \pm \frac{\sqrt{3}}{2}i$ です。初期値がどの根に収束するか、色分けすると面白い図ができます。

この図を描く C++ のプログラムは Form に PictureBox を貼り付け、サイズを 400 * 400 にします。



picturebox1 のイベント paint を次のように定義する。

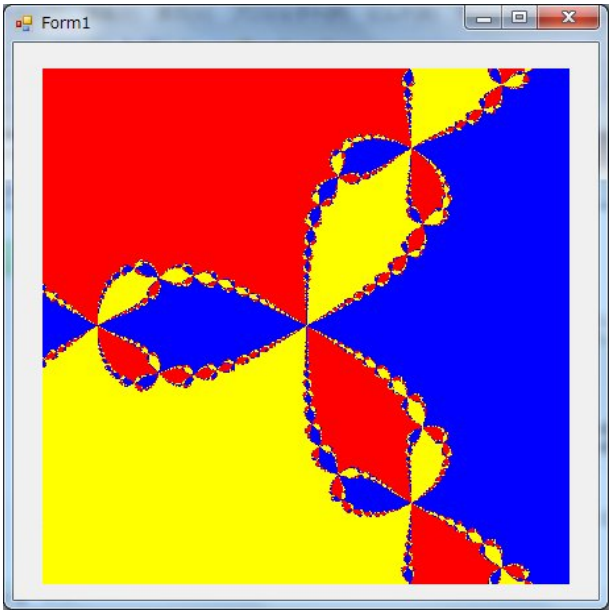
```
private: System::Void pictureBox1_Paint(System::Object^  sender,
                        System::Windows::Forms::PaintEventArgs^  e) {
```

```
        Graphics^ g = e->Graphics;
        Pen^ pen = gcnew Pen(Color::Blue);
        int x0 = 200, y0 = 200;
        int gx, gy;
        double limit = 0.000001;
        for (double x=-1; x<=1; x+=0.005) {
            for (double y=-1; y<=1; y+=0.005) {
                double a=x, b=y;
                while (fabs((a-1)*(a-1)+b*b)>limit
                    && fabs((a+0.5)*(a+0.5)+(b-sqrt(3.0)/2)*(b-sqrt(3.0)/2))>limit
                    && fabs((a+0.5)*(a+0.5)+(b+sqrt(3.0)/2)*(b+sqrt(3.0)/2))>limit) {
                    double c1 = a*a*a-3*a*b*b-1;
                    double c2 = 3*a*a*b-b*b*b;
                    double d1 = 3*(a*a-b*b);
                    double d2 = 6*a*b;
                    a = a - (c1*d1+c2*d2)/(d1*d1+d2*d2);
                    b = b - (c2*d1-c1*d2)/(d1*d1+d2*d2);
                }
                if (fabs((a-1)*(a-1)+b*b)<=limit)
                    pen = gcnew Pen(Color::Blue);
                else if (fabs((a+0.5)*(a+0.5)+(b-sqrt(3.0)/2)*(b-sqrt(3.0)/2))<=limit)
                    pen = gcnew Pen(Color::Red);
                else
                    pen = gcnew Pen(Color::Yellow);
                gx = (int)(x0 + 200*x);
                gy = (int)(y0 - 200*y);
                g->DrawLine(pen, gx, gy, gx+1, gy+1);
            }
        }
    }
```

そして、Form1.h の先頭に

```
#include <math.h>
```

と打ち込みます。
　実行すると



となります。

演習問題

1. 立方根の近似値を計算するプログラムを書け。
2. ニュートン法によって 1, 2, ... , 20 の平方根の近似値を計算し、結果を数表の形に見やすく出力するプログラムを書け。
3. 関数 $f(x) = x^2 - \cos x$ のグラフを描くプログラムを書け。

0.4 数値積分

定積分を数値的に求める方法が数学 C で説明されている。区分求積法、数値積分法として台形公式とシンプソンの公式である。区分求積法は定積分の定義であるが、普通は収束が速い台形公式やシンプソンの公式が使われる。$\int_a^b f(x)dx$ の値を近似的に計算することを考える。

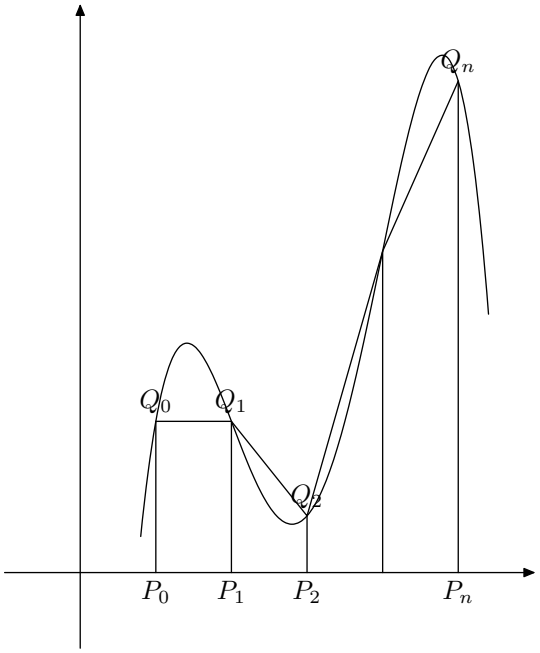
いま、区間 $[a, b]$ を n 等分して、幅が $h = (b-a)/n$ の小区間に分割する。そして、分点を小さい方から順に $P_0, P_1, P_2, \cdots, P_n$ とし、その x 座標の値を $a = x_0, x_1, x_2, \cdots, x_n = b$ とする。さらに、各々の分点 P_k における関数の値を $y_k = f(x_k)$ $(k = 0, 1, 2, \cdots, n)$ とし、座標 (x_k, y_k) の点を Q_k とする。曲線の弧 Q_kQ_{k+1} と三線分 $Q_kP_k, P_kP_{k+1}, P_{k+1}Q_{k+1}$ で囲まれる図形の面積を弧 Q_kQ_{k+1} を線分 Q_kQ_{k+1} で近似し、台形の面積

$$\frac{h}{2}(y_k + y_{k+1})$$

で近似する。

これらを加えると、つぎの公式が得られる。

$$\int_a^b f(x)dx \approx \frac{h}{2}\{y_0 + 2(y_1 + y_2 + \cdots + y_{n-1}) + y_n\}$$



次は台形公式公式による $\int_1^2 \frac{1}{x}dx$ の近似値を求める BASIC のプログラムである。

```
100 REM 台形公式
110 A=1:B=2
120 DEF FNY(X)=1/X
130 N=4
140 H=(B-A)/N
150 S=0
160 FOR K=1 TO N-1
170   S=S+FNY(A+K*H)
180 NEXT K
190 S=(FNY(A)+2*S+FNY(B))*H/2
200 PRINT "S=";S
210 PRINT "log(2)=";LOG(2)
220 END
```

実行すると

```
S= 0.6970238095238095
log(2)= 0.6931471805599453
```

です。

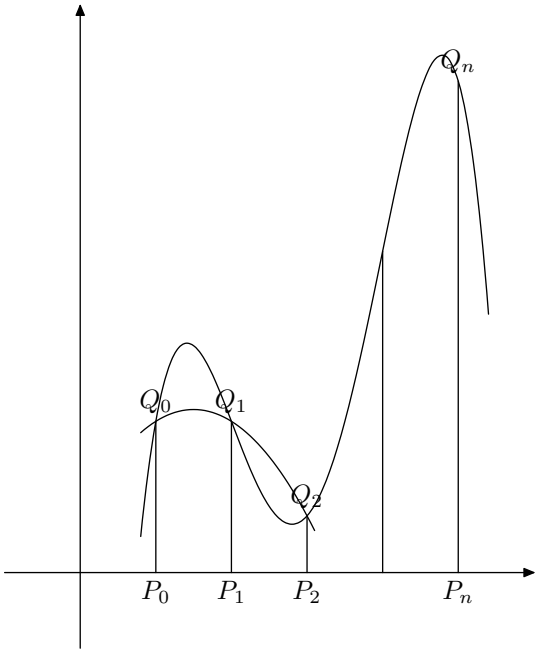
```
210 PRINT "log(2)=";LOG(2)
```

は、真の値と比較するために入れています。分割数 N が小さすぎるので、余り良い近似値ではないです。

上の BASIC のプログラムを C++ に翻訳すると次のようになります。

```
#include <iostream>
#include <stdio.h>
#include <math.h>

using namespace std;

double func(double x) {
    return 1.0/x;
}

int main()
{
    double a = 1, b = 2;
    int n = 4;
    double h = (b-a)/n;
    double s = 0;

    for (int k=1; k<n; k++)
        s = s + func(a+k*h);
    s = (func(a)+2*s+func(b))*h/2;
    cout << "s=" << s << endl;
    cout << "log(2)=" << log(2.0) << endl;
    getchar();
    return 0;
}
```

シンプソンの公式の公式は次のようにして求める。台形公式で n 等分していたのを今度は $2n$ 等分する。曲線 $y = f(x)$ の弧 $Q_{2k}Q_{2k+1}Q_{2k+2}$ の代わりに点 $Q_{2k}, Q_{2k+1}, Q_{2k+2}$ を通る放物線で近似し面積を求める。



$c = \frac{a+b}{2}$ とし、点 $(a, f(a)), (c, f(c)), (b, f(b))$ を通る放物線を $y = ux^2 + vx + w$ とする。

$$\begin{cases} f(a) = ua^2 + va + w \\ f(b) = ub^2 + vb + w \\ f(c) = uc^2 + vc + ww \end{cases}$$

が成り立つ。したがって、

$$\begin{aligned} w &= f(a) - ua^2 - va \\ v(b-a) &= f(b) - f(a) + u(a^2 - b^2) \\ u &= \frac{2}{(a-b)^2}(f(a) + f(b) - 2f(c)) \end{aligned}$$

が成り立つ。それらを次の式に代入する。

$$\begin{aligned} &\textstyle\int_a^b (ux^2 + vx + w)dx \\ &= (b-a)\{\tfrac{1}{3}u(b^2 + ab + a^2) + \tfrac{1}{2}v(b+a) + w\} \\ &= (b-a)\{\tfrac{1}{3}u(b^2 + ab + b^2) + \tfrac{1}{2}v(b+a) + f(a) - ua^2 - va\} \\ &= (b-a)\{\tfrac{1}{3}u(b^2 + ab - 2a^2) + \tfrac{1}{2}v(b-a) + f(a)\} \\ &= (b-a)\{\tfrac{1}{3}u(b^2 + ab - 2a^2) + \tfrac{1}{2}(f(b) - f(a) + u(a^2 - b^2)) + f(a)\} \\ &= (b-a)\{\tfrac{1}{3}\tfrac{2}{(a-b)^2}(f(a) + f(b) - 2f(c))\tfrac{-a^2+2ab-b^2}{2} + \tfrac{1}{2}(f(b) + f(a))\} \\ &= \tfrac{b-a}{6}(f(b) + f(a) + 4f(c)) \end{aligned}$$

したがって、 $h = (b-a)/(2n)$ とすれば、近似値は

$$\frac{h}{3}(y_{2k+2} + 4y_{2k+1} + y_{2k})$$

である。これらを加えると、次の近似公式がえられる。これをシンプソンの公式という。

$$\int_a^b f(x)dx \approx \frac{h}{3}\{y_0 + 4(y_1 + y_3 + \cdots + y_{2n-1}) + 2(y_2 + y_4 + \cdots + y_{2n-2}) + y_{2n}\}$$

つぎはシンプソンの公式による $\int_1^2 \frac{1}{x}dx$ の近似値を求める BASIC のプログラムである。

```
100 REM シンプソンの公式
110 A=1:B=2
120 DEF FNY(X)=1/X
130 N=4
140 H=(B-A)/N
150 S1=0:S2=0
160 FOR K=1 TO N-1 STEP 2
170   S1=S1+FNY(A+K*H):S2=S2+FNY(A+(K+1)*H)
180 NEXT K
190 S=(FNY(A)+4*S1+2*S2-FNY(B))*H/3
200 PRINT "S=";S
210 END
```

実行すると

```
S= 0.6931545306545307
```

となる。

上の BASIC のプログラムを C++ に翻訳すると次のようになります。

```
#include <iostream>
#include <stdio.h>
#include <math.h>

using namespace std;

double f(double x) {
    return 1.0/x;
}

int main()
{
    double a = 1, b = 2;
    double n = 4;
    double h = (b-a)/n;
    double s1 = 0, s2 = 0;

    for (int k=1; k<n; k += 2) {
        s1 = s1 + f(a+k*h);
        s2 = s2 + f(a+(k+1)*h);
    }
    double s = (f(a) + 4*s1 + 2*s2 - f(b))*h/3;
    cout << "s=" << s << endl;
```

```
    cout << "log(2)=" << log(2.0) << endl;
    getchar();
    return 0;
}
```

n の値を大きくすると良い近似が得られます。

演習問題

1. 定積分 $\int_0^1 \sqrt{1-x^2}dx$ を、区間の分割数が１０、２０、…、６０の場合についてシンプソンの公式で計算し、誤差を調べるプログラムを書け。ただし、この定積分の値は $\frac{\pi}{4}$ であり、有効桁数６桁で表すと０．７８５３９６である。

2. 放物線 $y = x^2$ の $x = 0$ から $x = 1$ までの弧長を求めるプログラムを書け。

0.5 連立一次方程式の解法

数学 C では連立一次方程式を解くガウスの消去法が説明され BASIC のプログラムが与えられている。つぎの BASIC のプログラムはガウスの消去法で連立一次方程式を解くものである。

```
100 REM ガウスの消去法
110 INPUT "N=";N
120 DIM A(N,N),B(N)
130 REM 方程式の入力
140 FOR I=1 TO N
150   FOR J=1 TO N
160     PRINT "A(";USING "#" I;USING "#" J;")=";:INPUT A(I,J)
170   NEXT J
180   PRINT "B(";USING "#" I;")=";:INPUT B(I)
190 NEXT I
200 REM 前進消去
210 FOR K=1 TO N-1
220   REM 行の入れ換え
230   AMAX=ABS(A(K,K)):KMAX=K
240   FOR I=K+1 TO N
250     IF ABS(A(I,K))>AMAX THEN KMAX=I:AMAX=ABS(A(I,K))
260   NEXT I
270   IF AMAX<0.00001 THEN PRINT "解けない":END
280   IF KMAX=K THEN 340
290   FOR J=K TO N
300     T=A(K,J):A(K,J)=A(KMAX,J):A(KMAX,J)=T
310   NEXT J
320   T=B(K):B(K)=B(KMAX):B(KMAX)=T
330   REM 前進消去の中心部
340   FOR I=K+1 TO N
350     A(I,K)=A(I,K)/A(K,K)
```

```
360     FOR J=K+1 TO N
370       A(I,J)=a(I,J)-A(I,K)*A(K,J)
380     NEXT J
390     B(I)=B(I)-A(I,K)*B(K)
400   NEXT I
410 NEXT K
420 IF ABS(A(N,N))<0.00001 THEN PRINT "解けない" : END
430 REM 後退代入
440 FOR K=N TO 1 STEP -1
450   FOR I=K+1 TO N
460     B(K)=B(K)-A(K,I)*B(I)
470   NEXT
480   B(K)=B(K)/A(K,K)
490 NEXT K
500 REM 解の出力
510 PRINT "X ";
520 FOR K=1 TO N
530   PRINT B(K);
540 NEXT K
550 END
```

実行すると

```
N=3
A( 1 1 )= ? 1
A( 1 2 )= ? 2
A( 1 3 )= ? -1
B( 1 )= ? -3
A( 2 1 )= ? 2
A( 2 2 )= ? 4
A( 2 3 )= ? 1
B( 2 )= ? 0
A( 3 1 )= ? 3
A( 3 2 )= ? 8
A( 3 3 )= ? 1
B( 3 )= ? -3
X  1.0000000000000007 -1.0000000000000002 2.0
```

となる。上の BASIC のプログラムを C++ に翻訳すると次のようになります。

```
#include "StdAfx.h"
#include <iostream>
#include <stdio.h>
#include <math.h>
```

```
using namespace std;

int main() {
    int n;
    cout << "n="; cin >> n;

    double **A, *B;
    A = new double*[n];
    for (int i=0; i<n; i++)
        A[i] = new double[n];
    B = new double[n];
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            cout << "A[" << i << "][" << j << "]="; cin >> A[i][j];
        }
        cout << "B[" << i << "]="; cin >> B[i];
    }
    for (int k=0; k<n; k++) {
        double AMAX = fabs(A[k][k]); int KMAX = k;
        for (int i=k+1; i<n; i++) {
            if (fabs(A[i][k]) > AMAX) {
                KMAX = i;
                AMAX = fabs(A[i][k]);
            }
        }
        if (AMAX < 0.00001) {
            cout << "解けない" << endl;
            for (int i=0; i<n; i++)
                delete A[i];
            delete A;
            getchar();
            getchar();
            return 1;
        }
        if (KMAX != k) {
            for (int j=k; j<n; j++) {
                double T = A[k][j];
                A[k][j]= A[KMAX][j];
                A[KMAX][j] = T;
            }
            double T = B[k];
            B[k] = B[KMAX];
            B[KMAX] = T;
```

```
        }
        for (int i=k+1; i<n; i++) {
            A[i][k] = A[i][k] / A[k][k];
            for (int j=k+1; j<n; j++) {
                A[i][j] = A[i][j] - A[i][k] * A[k][j];
            }
            B[i] = B[i] - A[i][k] * B[k];
        }
    }
    if (fabs(A[n-1][n-1]) < 0.00001) {
        cout << "解けない" << endl;
        for (int i=0; i<n; i++)
            delete A[i];
        delete A;
        delete B;
        getchar();
        getchar();
        return 1;
    }
    for (int k=n-1; k>=0; k--) {
        for (int i=k+1; i<n; i++) {
            B[k] = B[k] - A[k][i] * B[i];
        }
        B[k] = B[k] / A[k][k];
    }
    cout << " X ";
    for (int k=0; k<n; k++) {
        cout << B[k] << " ";
    }
    cout << endl;
    for (int i=0; i<n; i++)
        delete A[i];
    delete A;
    delete B;
    getchar();
    getchar();
    return 0;
}
```

n次正方行列の逆行列を求めるには連立一次方程式をn個同時に解けばよい。次は逆行列を求める BASIC プログラムである。

```
100 REM ガウスの消去法
110 INPUT "N=";N
```

```
120 DIM A(N,N),B(N,N)
130 REM 行列の入力
140 FOR I=1 TO N
150   FOR J=1 TO N
160     PRINT "A(";USING "#" I;USING "#" J;")=";:INPUT A(I,J)
170   IF I=J THEN B(I,J)=1 ELSE B(I,J)=0
180   NEXT J
190 NEXT I
200 REM 前進消去
210 FOR K=1 TO N-1
220   REM 行の入れ換え
230   AMAX=ABS(A(K,K)):KMAX=K
240   FOR I=K+1 TO N
250     IF ABS(A(I,K))>AMAX THEN KMAX=I:AMAX=ABS(A(I,K))
260   NEXT I
270   IF AMAX<0.00001 THEN PRINT "解けない":END
280   IF KMAX=K THEN 340
290   FOR J=K TO N
300     T=A(K,J):A(K,J)=A(KMAX,J):A(KMAX,J)=T
310   NEXT J
317   FOR J=1 TO N
320     T=B(K,J):B(K,J)=B(KMAX,J):B(KMAX,J)=T
323   NEXT J
330   REM 前進消去の中心部
340   FOR I=K+1 TO N
350     A(I,K)=A(I,K)/A(K,K)
360     FOR J=K+1 TO N
370       A(I,J)=a(I,J)-A(I,K)*A(K,J)
380     NEXT J
387     FOR J=1 TO N
390       B(I,J)=B(I,J)-A(I,K)*B(K,J)
393     NEXT J
400   NEXT I
410 NEXT K
420 IF ABS(A(N,N))<0.00001 THEN PRINT "解けない" : END
430 REM 後退代入
440 FOR K=N TO 1 STEP -1
450   FOR I=K+1 TO N
457     FOR J=1 TO N
460       B(K,J)=B(K,J)-A(K,I)*B(I,J)
463     NEXT J
470   NEXT
477   FOR J=1 TO N
```

```
480     B(K,J)=B(K,J)/A(K,K)
483   NEXT J
490 NEXT K
500 REM 解の出力
510 PRINT "逆行列 "
520 FOR K=1 TO N
527   FOR J=1 TO N
530     PRINT B(K,J);
533   NEXT J
536   PRINT
540 NEXT K
550 END
```

実行すると

```
N=2
A( 1 1 )= ? 2
A( 1 2 )= ? 1
A( 2 1 )= ? 3
A( 2 2 )= ? 4
逆行列
0.8000000000000002 -0.20000000000000004
-0.6000000000000001 0.4
```

となる。

上の BASIC のプログラムを C++ に翻訳すると次のようになります。

```
#include "StdAfx.h"
#include <iostream>
#include <stdio.h>
#include <math.h>

using namespace std;

int main() {
    int n;
    cout << "n="; cin >> n;

    double **A, **B;
    A = new double*[n];
    for (int i=0; i<n; i++)
        A[i] = new double[n];
    B = new double*[n];
    for (int i=0; i<n; i++)
        B[i] = new double[n];
```

```
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) {
        cout << "A[" << i << "][" << j << "]="; cin >> A[i][j];
        if (i == j)
            B[i][j] = 1;
        else
            B[i][j] = 0;
    }
}
for (int k=0; k<n; k++) {
    double AMAX = fabs(A[k][k]); int KMAX = k;
    for (int i=k+1; i<n; i++) {
        if (fabs(A[i][k]) > AMAX) {
            KMAX = i;
            AMAX = fabs(A[i][k]);
        }
    }
    if (AMAX < 0.00001) {
        cout << "解けない" << endl;
        for (int i=0; i<n; i++)
            delete A[i];
        delete A;
        for (int i=0; i<n; i++)
            delete B[i];
        delete B;
        getchar();
        getchar();
        return 1;
    }
    if (KMAX != k) {
        for (int j=k; j<n; j++) {
            double T = A[k][j];
            A[k][j]= A[KMAX][j];
            A[KMAX][j] = T;
        }
        for (int j=k; j<n; j++) {
            double T = B[k][j];
            B[k][j]= B[KMAX][j];
            B[KMAX][j] = T;
        }
    }
    for (int i=k+1; i<n; i++) {
        A[i][k] = A[i][k] / A[k][k];
```

```
        for (int j=k+1; j<n; j++) {
            A[i][j] = A[i][j] - A[i][k] * A[k][j];
        }
        for (int j=k+1; j<n; j++) {
            B[i][j] = B[i][j] - A[i][k] * B[k][j];
        }
    }
}
if (fabs(A[n-1][n-1]) < 0.00001) {
    cout << "解けない" << endl;
    for (int i=0; i<n; i++)
        delete A[i];
    delete A;
    for (int i=0; i<n; i++)
        delete B[i];
    delete B;
    getchar();
    getchar();
    return 1;
}
for (int k=n-1; k>=0; k--) {
    for (int i=k+1; i<n; i++) {
        for (int j=0; j<n; j++) {
            B[k][j] = B[k][j] - A[k][i] * B[i][j];
        }
    }
    for (int j=0; j<n; j++) {
        B[k][j] = B[k][j] / A[k][k];
    }
}
cout << " 逆行列： " << endl;
for (int k=0; k<n; k++) {
    for (int j=0; j<n; j++) {
        cout << B[k][j] << " ";
    }
    cout << endl;
}
for (int i=0; i<n; i++)
    delete A[i];
delete A;
for (int i=0; i<n; i++)
    delete B[i];
delete B;
```

```
    getchar();
    getchar();
    return 0;
}
```

演習問題

1. 基本的な考え方はガウスの消去法と同じであるが、前進消去の段階で、対角線より下にある成分だけでなく、対角線より上にある成分も同時に消去してしまう方法もある。これをガウス・ジョルダンの消去法あるいは掃出し法という。ガウス・ジョルダンの消去法により連立一次方程式を解くプログラムを書け。

挑戦問題

1. バイオリズムは、身体の周期が２３日、感情の周期が２８日、知性の周期が３３日で、次のように計算されます。

$$physical = \sin(T/23 * 2 * PI)$$
$$sensitivity = \sin(T/28 * 2 * PI)$$
$$intellectual = \sin(T/33 * 2 * PI)$$

ここで、T は誕生日から計算する日までの日数で、PI は円周率です。各々の値が正ならば好調、負ならば不調、零ならば好不調の変わり目の要注意日となります。ある日の前後のバイオリズム曲線を描く場合には、上の式で T をその日の前後に変化させて、値を線で結べばよいです。バイオリズムを表示するプログラムを作れ。

今後の勉強法

後は興味のある分野を勉強すればいいです。人が面白そうなことをしているのを見れば、真似をしてやってみれば良いです。コンピュータ囲碁のプログラム「アルファ碁」が韓国のプロ棋士イ・セドンさんに 4 勝 1 敗で勝ちました。「深層学習」によって強くなったと書かれています。人工知能の最近の研究では「機械学習」や最新の「深層学習」や古くは「ニューラルネットワーク」や「遺伝的アルゴリズム」や「遺伝的プログラミング」などコンピュータに学習させることが主流になっています。いきなり「深層学習」を勉強するのは難しいので、元になった「ニューラルネットワーク」を勉強しようと思ったとします。このような場合、私のような素人はいきなり理論から勉強するより、プログラムリストの載っている平野廣美著「C でつくるニューラルネットワーク」パーソナルメディアのような本を選ぶのが良いです。ただこの本は１９９１年に初版が出ている古い本で、使われている C は MS-C Ver.5.1 のもので、初期の文法に基づいて書かれていますし、第 2 章以降で使われているグラフィックスは今では通用しないものです。私のように日本に C が紹介されてからの歴史を知っているものにとっても、読みこなすのはしんどいです。ただ、この本には、エジンバラ大学の生物学教授 Donald Michie 先生が、マッチ箱とビーズが、人間相手に Tic Tac Too を行い、勝つための方策を学習する仕組みを考案したことがまず紹介されています。そして、コンピュータがあれば、この仕組みを実現するプログラムを簡単に作ることが出来ると書いてあ

りますがそのプログラムリストは載っていません。代わりに、「マッチ箱がおしどりゲームをマスターする」として、「おしどりゲーム」というパズルが紹介されていますが、この為のプログラムリストも与えられていません。この問題を抽象化した、各点の分岐が３で、深さ３の木を考え、その葉（端点）の１つがゴールである状況での探索を学習するプログラムのリストが載っています。具体的な問題では、その問題固有の厄介なことが出てきますが、このように抽象化し、単純化すると物事の本質が見えるようになり、プログラムも簡単になり、始めてこの分野を勉強する人にも理解しやすくなります。

```
struct { int ncase;
int beans[3];
int next[3];
} match_box[40] = {
{ 3, { Beans, Beans, Beans }, {  1,  2,  3 } },
{ 3, { Beans, Beans, Beans }, {  4,  5,  6 } },
{ 3, { Beans, Beans, Beans }, {  7,  8,  9 } },
{ 3, { Beans, Beans, Beans }, { 10, 11, 12 } },
{ 3, { Beans, Beans, Beans }, { 13, 14, 15 } },
{ 3, { Beans, Beans, Beans }, { 16, 17, 18 } },
{ 3, { Beans, Beans, Beans }, { 19, 20, 21 } },
{ 3, { Beans, Beans, Beans }, { 22, 23, 24 } },
{ 3, { Beans, Beans, Beans }, { 25, 26, 27 } },
{ 3, { Beans, Beans, Beans }, { 28, 29, 30 } },
{ 3, { Beans, Beans, Beans }, { 31, 32, 33 } },
{ 3, { Beans, Beans, Beans }, { 34, 35, 36 } },
{ 3, { Beans, Beans, Beans }, { 37, 38, 39 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
```

```
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } },
{ 0, { Beans, Beans, Beans }, { -1, -1, -1 } }
};
```

で、マッチ箱を表現しています。構造体の配列が使われています。複雑なプログラムを作るためには、配列やスタックやキューなどのデータ構造を勉強しなければなりません。そしてこれらのデータ構造を扱うためのアルゴリズムや探索などのアルゴリズムの勉強も必要です。昔は Niklaus Wirth 著「アルゴリズム＋データ構造＝プログラム」という素晴らしい参考書がありましたが今は絶版ですし、今は使われていない PASCAL で書かれています。

抽象化し、単純化したので、このようにマッチ箱のシステムを人間が作ることが出来ます。そして、このシステムで学習するプログラムのプログラムリストが載っているので、実際にコンピュータで動かしたり、プログラムリストを読んでみることにより、本に書いていることを理解することが出来ます。本ではこの本の主題である「ニューラルネットワーク」の説明に移ります。才能のある人はこれで全てを理解すると思いますが、普通の我々凡人はここで止めてはいけません。コンピュータがあれば、「マッチ箱とビーズが、人間相手に Tic Tac Too を行い、勝つための方策を学習する」仕組みを実現するプログラムを簡単に作ることが出来るに納得できれば、目をつぶればプログラムリストのイメージが頭の中に浮かんで来れば、先に進んで良いですが、そうでなければ自分でこのプログラムを与えられたこの本のプログラムリストをお手本に、真似をして作ってみます。数学者がこの定理はトリビアルであると言ってその証明を書いていないときは、「本当に簡単である場合」と「本当はめんどくさくて書くのが厄介だが、やろうと思えば自分は出来るという主張である場合」と「実は間違っている場合」とがあるそうです。Tic Tac Too の問題も結構厄介です。この本の作者もやろうと思えば自分は出来るよと主張しています。こんな問題を一つ一つ解いてみることが、将来、大きな違いになってきます。勉強はこの本に載っているプログラムリストを丸暗記するとこではなく、自分でそのようなリストを何もないところから書けるようになることです。問題はマッチ箱のシステムを作る部分で、そのシステムを使って学習する部分は本のプログラムリストを真似すれば良いです。Tic Tac Too のマッチ箱のシステムをサンプルプログラムのように人間が一つ一つ書いていくことは出来ません。ざっと計算してみると、最初は 9 か所○を書く場所があります。次は 8 か所×を書く場所があります。次に七か所○を書く場所があります。このように考えると途中で勝負がつく場合がありますが全体で約 9! 個のマッチ箱が必要です。膨大な個数です。対称性を考慮すれば大幅に数を減らせます。コンピュータにマッチ箱システムを作らせなければなりません。

```
struct Match_Box {
    int result;
    int teban;
    int ncase;
    int beans[9];
```

```
    int ban[9];
    int next[9];
};
vector<Match_Box> match_box;
```

のようなマッチ箱を考えます。木の深さが一定でないので、result で勝ち、負け、引き分け、途中の区別を示します。teban で○の手番か×の手番かを示します。 ban[9] で現在の盤面を示します。ncase で分岐の個数を示します。beans[9] でビーズの個数を示します。next[9] で分岐先を示します。マッチ箱が何個いるか分からないので配列ではなく vector を使います。

Tic Tac Too のマッチ箱のシステムをコンピュータに作らせるプログラムは次のようになります。

```
#include "stdafx.h"
#include <iostream>
#include <math.h>
#include <vector>
#include <queue>

using namespace std;

const int MARU =  1;
const int BATU = -1;

#define Beans 20
#define MaxLevel 9
#define Learn 0
#define Test 1
#define Prize 3

#define rnd() { (double)rand()/0x7fff }

struct Match_Box {
int result;
int teban;
int ncase;
int beans[9];
int ban[9];
int next[9];
};
vector<Match_Box> match_box;

int board[9];

bool terninalP(int &result) {
```

```
// board[9] の情報で勝敗がついていれば、return true
// ○の勝ち
if (board[0] == 1 && board[1] == 1 && board[2] == 1) {
    result = 1;
    return true;
}
if (board[0] == 1 && board[3] == 1 && board[6] == 1) {
    result = 1;
   return true;
}
if (board[0] == 1 && board[4] == 1 && board[8] == 1) {
    result = 1;
    return true;
}
if (board[1] == 1 && board[4] == 1 && board[7] == 1) {
    result = 1;
    return true;
}
if (board[2] == 1 && board[5] == 1 && board[8] == 1) {
    result = 1;
    return true;
}
if (board[2] == 1 && board[4] == 1 && board[6] == 1) {
    result = 1;
    return true;
}
if (board[3] == 1 && board[4] == 1 && board[5] == 1) {
    result = 1;
    return true;
}
if (board[6] == 1 && board[7] == 1 && board[8] == 1) {
    result = 1;
    return true;
}
// ×の勝ち
if (board[0] == -1 && board[1] == -1 && board[2] == -1) {
    result = -1;
    return true;
}
if (board[0] == -1 && board[3] == -1 && board[6] == -1) {
    result = -1;
    return true;
}
```

```
if (board[0] == -1 && board[4] == -1 && board[8] == -1) {
    result = -1;
    return true;
}
if (board[1] == -1 && board[4] == -1 && board[7] == -1) {
    result = -1;
    return true;
}
if (board[2] == -1 && board[5] == -1 && board[8] == -1) {
    result = -1;
    return true;
}
if (board[2] == -1 && board[4] == -1 && board[6] == -1) {
    result = -1;
    return true;
}
if (board[3] == -1 && board[4] == -1 && board[5] == -1) {
    result = -1;
    return true;
}
if (board[6] == -1 && board[7] == -1 && board[8] == -1) {
    result = -1;
    return true;
}
// 引き分け
int count = 0;
while (true) {
    int maru = 0;
    int batu = 0;
    if (board[0] == 1) maru++;
    else if (board[0] == -1) batu++;
    if (board[3] == 1) maru++;
    else if (board[3] == -1) batu++;
    if (board[6] == 1) maru++;
    else if (board[6] == -1) batu++;
    if (maru == 0 || batu == 0) {
        result = 9;
        return false;
    }
    maru = 0;
    batu = 0;
    if (board[0] == 1) maru++;
    else if (board[0] == -1) batu++;
```

```
if (board[1] == 1) maru++;
else if (board[1] == -1) batu++;
if (board[2] == 1) maru++;
else if (board[2] == -1) batu++;
if (maru == 0 || batu == 0) {
    result = 9;
    return false;
}
maru = 0;
batu = 0;
if (board[0] == 1) maru++;
else if (board[0] == -1) batu++;
if (board[4] == 1) maru++;
else if (board[4] == -1) batu++;
if (board[8] == 1) maru++;
else if (board[8] == -1) batu++;
if (maru == 0 || batu == 0) {
    result = 9;
    return false;
}
maru = 0;
batu = 0;
if (board[1] == 1) maru++;
else if (board[1] == -1) batu++;
if (board[4] == 1) maru++;
else if (board[4] == -1) batu++;
if (board[7] == 1) maru++;
else if (board[7] == -1) batu++;
if (maru == 0 || batu == 0) {
    result = 9;
    return false;
}
maru = 0;
batu = 0;
if (board[2] == 1) maru++;
else if (board[2] == -1) batu++;
if (board[4] == 1) maru++;
else if (board[4] == -1) batu++;
if (board[6] == 1) maru++;
else if (board[6] == -1) batu++;
if (maru == 0 || batu == 0) {
    result = 9;
    return false;
```

```
        }
        maru = 0;
        batu = 0;
        if (board[2] == 1) maru++;
        else if (board[2] == -1) batu++;
        if (board[5] == 1) maru++;
        else if (board[5] == -1) batu++;
        if (board[8] == 1) maru++;
        else if (board[8] == -1) batu++;
        if (maru == 0 || batu == 0) {
            result = 9;
            return false;
        }
        maru = 0;
        batu = 0;
        if (board[3] == 1) maru++;
        else if (board[3] == -1) batu++;
        if (board[4] == 1) maru++;
        else if (board[4] == -1) batu++;
        if (board[5] == 1) maru++;
        else if (board[5] == -1) batu++;
        if (maru == 0 || batu == 0) {
            result = 9;
            return false;
        }
        maru = 0;
        batu = 0;
        if (board[6] == 1) maru++;
        else if (board[6] == -1) batu++;
        if (board[7] == 1) maru++;
        else if (board[7] == -1) batu++;
        if (board[8] == 1) maru++;
        else if (board[8] == -1) batu++;
        if (maru == 0 || batu == 0) {
            result = 9;
            return false;
        }
        break;
    }
    result = 0;
    return true;
}
```

```
void getNextCandidate(Match_Box Box, vector<Match_Box> &cand) {
    Match_Box newBox;
    for (int k = 0; k < 9; k++) {
        if (Box.ban[k] != 0) continue;
        newBox = Box;
        if (Box.teban == MARU) {
            newBox.teban = BATU;
            newBox.ban[k] = MARU;
        }
        else {
            newBox.teban = MARU;
            newBox.ban[k] = BATU;
        }
        // newBox.ban[] は新しいか？
        bool flag = false;
        // そのまま
        int tban[9];
        for (int j = 0; j < 9; j++)
            tban[j] = newBox.ban[j];
        for (int i = 0; i < cand.size(); i++) {
            bool flag2 = true;
            for (int j = 0; j < 9; j++) {
                if (tban[j] != cand[i].ban[j]) {
                    flag2 = false; break;
                }
            }
            if (flag2) { // 一致した
                flag = true; break;
            }
        }
        if (!flag) {
            // ９０度回転
            tban[0] = newBox.ban[2];
            tban[1] = newBox.ban[5];
            tban[2] = newBox.ban[8];
            tban[3] = newBox.ban[1];
            tban[4] = newBox.ban[4];
            tban[5] = newBox.ban[7];
            tban[6] = newBox.ban[0];
            tban[7] = newBox.ban[3];
            tban[8] = newBox.ban[6];
            for (int i = 0; i < cand.size(); i++) {
                bool flag2 = true;
```

```
        for (int j = 0; j < 9; j++) {
            if (tban[j] != cand[i].ban[j]) {
                flag2 = false; break;
            }
        }
        if (flag2) { // 一致した
            flag = true; break;
        }
    }
}
if (!flag) {
    // １８０度回転
    tban[0] = newBox.ban[8];
    tban[1] = newBox.ban[7];
    tban[2] = newBox.ban[6];
    tban[3] = newBox.ban[5];
    tban[4] = newBox.ban[4];
    tban[5] = newBox.ban[3];
    tban[6] = newBox.ban[2];
    tban[7] = newBox.ban[1];
    tban[8] = newBox.ban[0];
    for (int i = 0; i < cand.size(); i++) {
        bool flag2 = true;
        for (int j = 0; j < 9; j++) {
            if (tban[j] != cand[i].ban[j]) {
                flag2 = false; break;
            }
        }
        if (flag2) { // 一致した
            flag = true; break;
        }
    }
}
if (!flag) {
    // ２７０度回転
    tban[0] = newBox.ban[6];
    tban[1] = newBox.ban[3];
    tban[2] = newBox.ban[0];
    tban[3] = newBox.ban[7];
    tban[4] = newBox.ban[4];
    tban[5] = newBox.ban[1];
    tban[6] = newBox.ban[8];
    tban[7] = newBox.ban[5];
```

```
    tban[8] = newBox.ban[2];
    for (int i = 0; i < cand.size(); i++) {
        bool flag2 = true;
        for (int j = 0; j < 9; j++) {
            if (tban[j] != cand[i].ban[j]) {
                flag2 = false; break;
            }
        }
        if (flag2) { // 一致した
            flag = true; break;
        }
    }
}
if (!flag) {
    // 線対称
    tban[0] = newBox.ban[6];
    tban[1] = newBox.ban[7];
    tban[2] = newBox.ban[8];
    tban[3] = newBox.ban[3];
    tban[4] = newBox.ban[4];
    tban[5] = newBox.ban[5];
    tban[6] = newBox.ban[0];
    tban[7] = newBox.ban[1];
    tban[8] = newBox.ban[2];
    for (int i = 0; i < cand.size(); i++) {
        bool flag2 = true;
        for (int j = 0; j < 9; j++) {
            if (tban[j] != cand[i].ban[j]) {
                flag2 = false; break;
            }
        }
        if (flag2) { // 一致した
            flag = true; break;
        }
    }
}
if (!flag) {
    // 線対称：９０度回転
    tban[0] = newBox.ban[8];
    tban[1] = newBox.ban[5];
    tban[2] = newBox.ban[2];
    tban[3] = newBox.ban[7];
    tban[4] = newBox.ban[4];
```

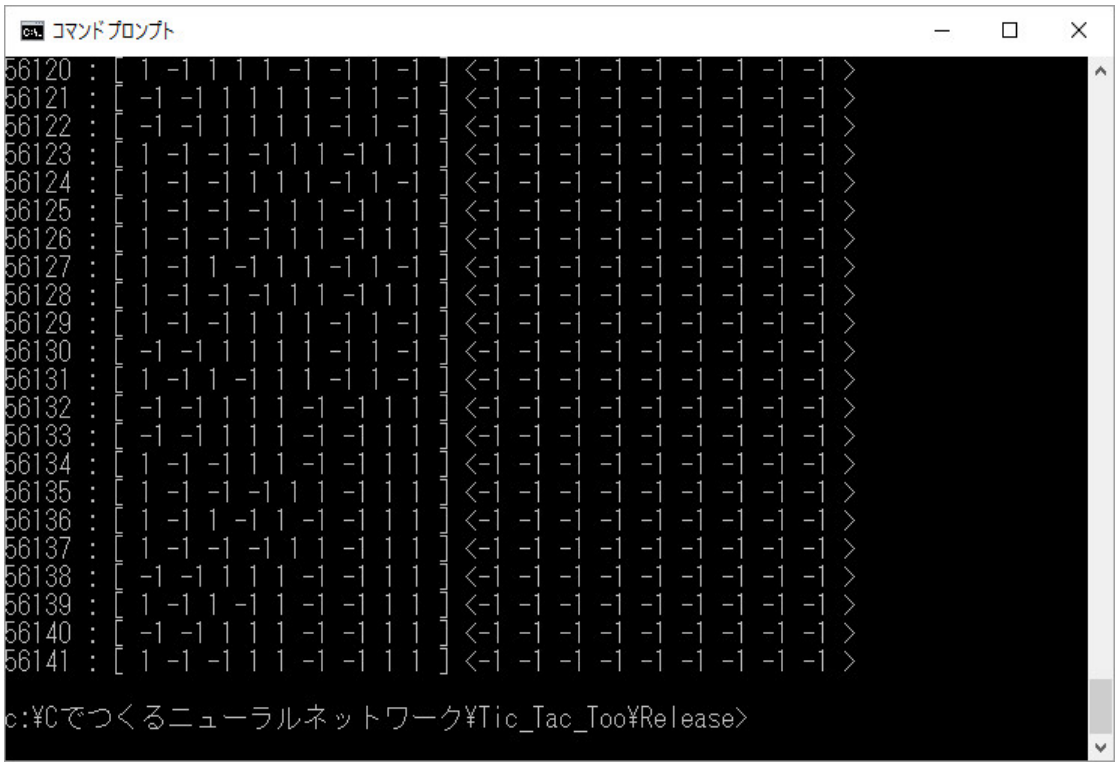
```
    tban[5] = newBox.ban[1];
    tban[6] = newBox.ban[6];
    tban[7] = newBox.ban[3];
    tban[8] = newBox.ban[0];
    for (int i = 0; i < cand.size(); i++) {
        bool flag2 = true;
        for (int j = 0; j < 9; j++) {
            if (tban[j] != cand[i].ban[j]) {
                flag2 = false; break;
            }
        }
        if (flag2) { // 一致した
             flag = true; break;
        }
    }
}
if (!flag) {
    // 線対称：１８０度回転
    tban[0] = newBox.ban[2];
    tban[1] = newBox.ban[1];
    tban[2] = newBox.ban[0];
    tban[3] = newBox.ban[5];
    tban[4] = newBox.ban[4];
    tban[5] = newBox.ban[3];
    tban[6] = newBox.ban[8];
    tban[7] = newBox.ban[7];
    tban[8] = newBox.ban[6];
    for (int i = 0; i < cand.size(); i++) {
        bool flag2 = true;
        for (int j = 0; j < 9; j++) {
            if (tban[j] != cand[i].ban[j]) {
                flag2 = false; break;
            }
        }
        if (flag2) { // 一致した
            flag = true; break;
        }
    }
}
if (!flag) {
    // 線対称：２７０度回転
    tban[0] = newBox.ban[0];
    tban[1] = newBox.ban[3];
```

```
            tban[2] = newBox.ban[6];
            tban[3] = newBox.ban[1];
            tban[4] = newBox.ban[4];
            tban[5] = newBox.ban[7];
            tban[6] = newBox.ban[2];
            tban[7] = newBox.ban[5];
            tban[8] = newBox.ban[8];
            for (int i = 0; i < cand.size(); i++) {
                bool flag2 = true;
                for (int j = 0; j < 9; j++) {
                    if (tban[j] != cand[i].ban[j]) {
                        flag2 = false; break;
                    }
                }
                if (flag2) { // 一致した
                    flag = true; break;
                }
            }
        }
        if (!flag) {
            cand.push_back(newBox);
        }
    }
}

void getMatchBox(vector<Match_Box> &match_box) {
    queue<Match_Box> qu;
    Match_Box Box;

    Box.teban = MARU; // 次の手番
    Box.result = 9; // 結果不明
    Box.ncase = 9; // 次のボックス番号の個数
    for (int i = 0; i < 9; i++) {
        Box.beans[i] = Beans;
        Box.next[i] = -1; // 次のボックス番号
        Box.ban[i] = 0;
    }
    int child_index = 1;
    qu.push(Box);
    while (!qu.empty()) {
        int index = match_box.size();
        // if (index > 16) break;
        Box = qu.front();
```

```
        qu.pop();
        if (Box.result == 1 || Box.result == -1 || Box.result == 0) {
            Box.ncase = 0;
            match_box.push_back(Box);
        }
        else {
            Match_Box newBox;
            // 対称でない次の候補を求める
            vector<Match_Box> cand;
            getNextCandidate(Box, cand);
            Box.ncase = cand.size();
            for (int k = 0; k < cand.size(); k++) {
                Box.next[k] = child_index++;
            }
            match_box.push_back(Box);
            for (int k = 0; k < cand.size(); k++) {
                newBox = cand[k];
                int result = 9;
                for (int j = 0; j < 9; j++) {
                    board[j] = newBox.ban[j];
                }
                if (terninalP(result)) { // 端点
                    newBox.result = result;
                    // newBox.ncase = 0;
                    for (int i = 0; i < 9; i++) {
                        newBox.beans[i] = Beans;
                        newBox.next[i] = -1;
                    }
                    qu.push(newBox);
                }
                else {
                    newBox.result = result;
                    for (int i = 0; i < 9; i++) {
                        newBox.beans[i] = Beans;
                        newBox.next[i] = -1;
                    }
                    qu.push(newBox);
                }
            }
        }
    }
}
```

```
int main()
{
    // match_box[] を作る
    getMatchBox(match_box);

    for (int i = 0; i < match_box.size(); i++) {
        cout << i << " : " << "[ ";
        for (int k = 0; k < 9; k++) {
             cout << match_box[i].ban[k] << " ";
        }
        cout << "] <";
        for (int k = 0; k < 9; k++) {
            cout << match_box[i].next[k] << " ";
        }
        cout << ">" << endl;
    }
    return 0
}
```

がマッチ箱のシステムを作るプログラムです。


コマンドプロンプト
56120 : [1 -1 1 1 1 -1 -1 1 -1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56121 : [-1 -1 1 1 1 1 -1 1 -1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56122 : [-1 -1 1 1 1 1 -1 1 -1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56123 : [1 -1 -1 -1 1 1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56124 : [1 -1 -1 1 1 1 -1 1 -1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56125 : [1 -1 -1 -1 1 1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56126 : [1 -1 -1 -1 1 1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56127 : [1 -1 1 -1 1 1 -1 1 -1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56128 : [1 -1 -1 -1 1 1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56129 : [1 -1 -1 1 1 1 -1 1 -1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56130 : [-1 -1 1 1 1 1 -1 1 -1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56131 : [1 -1 1 -1 1 1 -1 1 -1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56132 : [-1 -1 1 1 1 -1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56133 : [-1 -1 1 1 1 -1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56134 : [1 -1 -1 1 1 -1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56135 : [1 -1 -1 -1 1 1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56136 : [1 -1 1 -1 1 -1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56137 : [1 -1 -1 -1 1 1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56138 : [-1 -1 1 1 1 -1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56139 : [1 -1 1 -1 1 -1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56140 : [-1 -1 1 1 1 -1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
56141 : [1 -1 -1 1 1 -1 -1 1 1] <-1 -1 -1 -1 -1 -1 -1 -1 -1 >
c:¥Cでつくるニューラルネットワーク¥Tic_Tac_Too¥Release>
```

56141 個のマッチ箱を作っています。

後は本のプログラムを参考にし、学習と検証の部分を作ればいいです。学習するのは先手の○の手番とし、相手の×はランダムに手を選ぶ場合とマッチ箱を使って学習する場合と正確に読んで勝
```

てるときは勝てる手を選択する場合を選択できるようにします。更に、両方ともランダムに手を選んだ場合の勝率が分かるようにします。ただし、共に最善手を選べば引き分けになりますから、勝ちか引き分けの割合を求めます。プログラムの残りは

```
int select[MaxLevel + 1], nsselect[MaxLevel + 1];
int level, lcount, goodcount;

int tictactoo(int level) {
    // 端点か？
    int val = 9;
    if (terninalP(val)) {
        //  端点なら評価値を返す（MARU 勝ち：1, MARU 勝ち：-1, 引き分け：0）
        return val;
    }
    // 可能な手の集合を求める
    vector<int> telist;
    for (int i = 0; i < 9; i++) {
        if (board[i] == 0) {
            telist.push_back(i);
        }
    }

    if (level % 2 == 0) { // MARU の手番
        val = -100;
        for (int k = 0; k < telist.size(); k++) { // 最大の手を選択する
            board[telist[k]] = 1;
            int v = tictactoo(level + 1);
            if (v > val) {
                val = v;
            }
            board[telist[k]] = 0;
        }
        return val;
    }
    else { // BATU の手番
        val = 100;
        for (int k = 0; k < telist.size(); k++) { // 最小の手を選択する
            board[telist[k]] = -1;
            int v = tictactoo(level + 1);
            if (v < val) {
                val = v;
            }
            board[telist[k]] = 0;
```

```
        }
        return val;
    }
}

int anti_tictactoo(int level) {
    // 端点か？
    int val = 9;
    if (terninalP(val)) { //  端点なら評価値を返す（MARU勝ち：1，MARU勝ち：-1，引き分け：0）
        return val;
    }
    // 可能な手の集合を求める
    vector<int> telist;
    for (int i = 0; i < 9; i++) {
        if (board[i] == 0) {
            telist.push_back(i);
        }
    }

    if (level % 2 == 0) { // MARU の手番
        val = -100;
        for (int k = 0; k < telist.size(); k++) { // 最大の手を選択する
            board[telist[k]] = 1;
            int v = anti_tictactoo(level + 1);
            if (v > val) {
                val = v;
            }
            board[telist[k]] = 0;
        }
        return val;
    }
    else { // BATU の手番
        val = -100;
        for (int k = 0; k < telist.size(); k++) { // 最大の手を選択する
            board[telist[k]] = -1;
            int v = anti_tictactoo(level + 1);
            if (v > val) {
                val = v;
            }
            board[telist[k]] = 0;
        }
        return val;
```

```
    }
}

int next_select(int pmatch, int type) {
    int i;
    double r, b, a;

    if (match_box[pmatch].teban == MARU) {
        if (type != 4) { // マッチ箱により手を選ぶ
            r = rnd(); b = a = 0;
            for (i = 0; i < match_box[pmatch].ncase; i++)
                b += match_box[pmatch].beans[i];
            for (i = 0; i < match_box[pmatch].ncase; i++) {
                a += match_box[pmatch].beans[i];
                if (b != 0) {
                    if (r < a / b) return i;
                }
            }
            return i - 1;
        }
        else { // ランダムに手を選ぶ
            int index = rand() % match_box[pmatch].ncase;
            return index;
        }
    }
    else { // コンピュータの手を選ぶ
        if (type == 0) { // 最悪手を選ぶ
            Match_Box Box = match_box[pmatch];
            if (Box.ncase == 1) {
                return 0;
            }
            int MaxVal = -100;
            int MaxID = 0;
            for (i = 0; i < Box.ncase; i++) {
                for (int k = 0; k < 9; k++) {
                    board[k] = match_box[Box.next[i]].ban[k];
                }
                int val = anti_tictactoo(0);
                if (val > MaxVal) {
                    MaxVal = val;
                    MaxID = i;
                }
            }
```

```
            return MaxID;
        }
        else if (type == 1 || type == 4) { // ランダムに手を選ぶ
            int index = rand() % match_box[pmatch].ncase;
            return index;
        }
        else if (type == 2) { // マッチ箱により手を選ぶ
             r = rnd(); b = a = 0;
            for (i = 0; i < match_box[pmatch].ncase; i++)
                b += match_box[pmatch].beans[i];
            for (i = 0; i < match_box[pmatch].ncase; i++) {
                a += match_box[pmatch].beans[i];
                if (b != 0) {
                    if (r < a / b) return i;
                }
            }
            return i - 1;
        }
        else { // 最善手を選ぶ
            Match_Box Box = match_box[pmatch];
            if (Box.ncase == 1) {
                return 0;
            }
            int MinVal = 100;
            int MinID = 0;
            for (i = 0; i < Box.ncase; i++) {
                for (int k = 0; k < 9; k++) {
                    board[k] = match_box[Box.next[i]].ban[k];
                }
                int val = tictactoo(0);
                if (val < MinVal) {
                    MinVal = val;
                    MinID = i;
                }
            }
            return MinID;
        }
    }
}

int execute(int type) {
    int lv, ns;
```

```
    select[0] = nsselect[0] = 0;
    for (lv = 1; lv <= MaxLevel; lv++) {
        ns = next_select(select[lv - 1], type);
        if (match_box[select[lv - 1]].beans[ns] - 1 < 0) {
            printf(" Error\n"); break;
        }
        select[lv] = match_box[select[lv - 1]].next[ns];
        nsselect[lv] = ns;
        if (match_box[select[lv]].ncase == 0) {
            break;
        }
    }
    return lv + 1;
}

int execute2(int type) {
    int lv, ns;

    select[0] = nsselect[0] = 0;
    for (lv = 1; lv <= MaxLevel; lv++) {
        if (type != 4) {
            ns = next_select(select[lv - 1], 3);
        }
        else {
            ns = next_select(select[lv - 1], 4);
        }
        if (match_box[select[lv - 1]].beans[ns] - 1 < 0) {
            printf(" Error\n"); break;
        }
        select[lv] = match_box[select[lv - 1]].next[ns];
        nsselect[lv] = ns;
        if (match_box[select[lv]].ncase == 0) {
            break;
        }
    }
    return lv + 1;
}

void learn(int len, int type) {
    int lv;
    int result = match_box[select[len - 1]].result;
    if (result == 1) { // MARU が勝った
        for (lv = 0; lv < len; lv++) {
```

```
            if (match_box[select[lv]].teban == MARU) {
                match_box[select[lv]].beans[nsselect[lv + 1]] += Prize;
            }
            else if(type == 2) {
                match_box[select[lv]].beans[nsselect[lv + 1]]--;
            }
        }
    }
    else if (result == -1) { // BATU が勝った
        for (lv = 0; lv < len; lv++) {
            if (match_box[select[lv]].teban == MARU) {
                match_box[select[lv]].beans[nsselect[lv + 1]]--;
            }
            else if (type == 2) {
                match_box[select[lv]].beans[nsselect[lv + 1]] += Prize;
            }
        }
    }
    else { // 引き分け
        for (lv = 0; lv < len; lv++) {
            if (match_box[select[lv]].teban == MARU) {
                match_box[select[lv]].beans[nsselect[lv + 1]] += Prize;
            }
            else if (type == 2) {
                match_box[select[lv]].beans[nsselect[lv + 1]] += Prize;
            }
        }
    }
}

void print_path(int mode, int cycle, int len) {
     if (mode == Learn) {
         printf(" Learning cycle[");
     }
     else {
         printf(" Test     cycle[");
     }
    printf("%3d]", cycle);
    printf("  (1-%2d)", select[1]);
    for (int k = 2; k < len; k++) {
        printf(" -> (%d-%2d)", k, select[k]);
    }
    printf("\n");
```

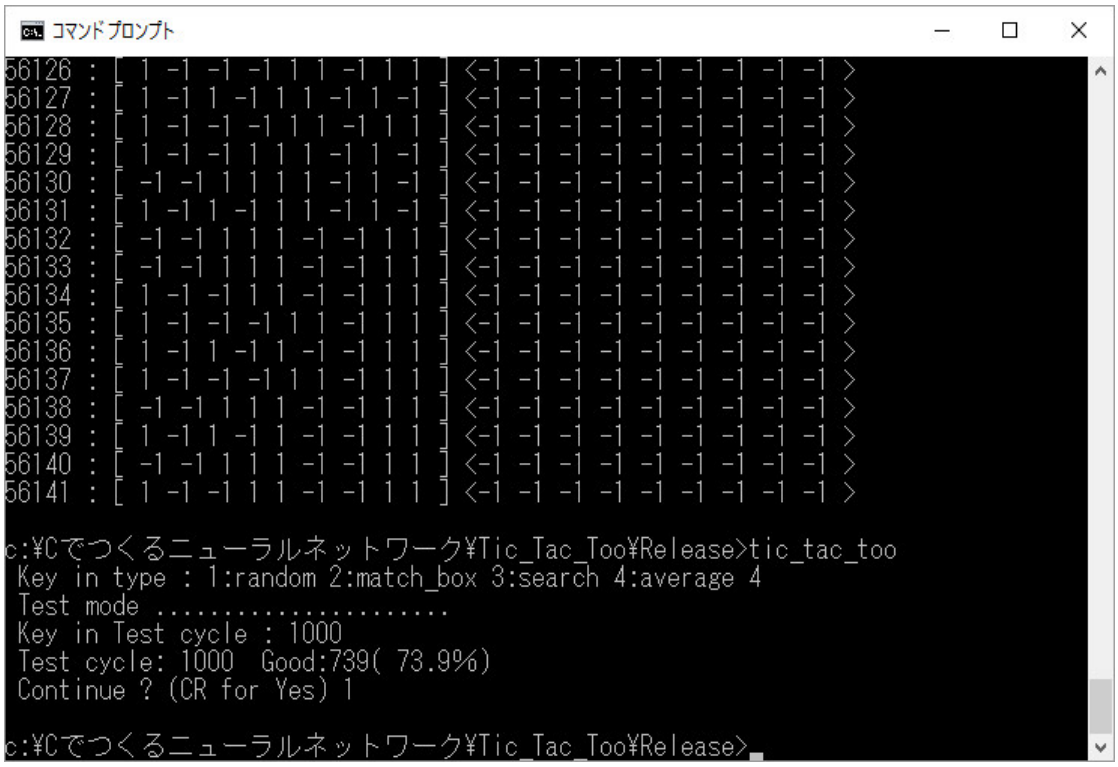
```
}

void print_result(int tcycle, int type) {
    double rate;

    rate = (100.0 * goodcount) / tcycle;
    if (type != 4) {
        printf(" Learning cycle: %d  Test cycle: %d  Good:%d(%5.1f %)\n",
            lcount, tcycle, goodcount, rate);
    }
    else {
        printf(" Test cycle: %d  Good:%d(%5.1f %)\n", tcycle, goodcount, rate);
    }
}

int main()
{
    // match_box[] を作る
    getMatchBox(match_box);
    /*
    for (int i = 0; i < match_box.size(); i++) {
        cout << i << " : " << "[ ";
        for (int k = 0; k < 9; k++) {
            cout << match_box[i].ban[k] << " ";
        }
        cout << "] <";
        for (int k = 0; k < 9; k++) {
            cout << match_box[i].next[k] << " ";
        }
        cout << ">" << endl;
    }
    */
    /*
    int P[10];
    P[0] = 3; P[1] = 14; P[2] = 76; P[3] = 414; P[4] = 2022;
    P[5] = 7709; P[6] = 22834; P[7] = 43671;
    for (int i = 0; i < 8; i++) {
        cout << P[i] << " : " << "[ ";
        for (int k = 0; k < 9; k++) {
            cout << match_box[P[i]].ban[k] << " ";
        }
        cout << "] <";
        for (int k = 0; k < 9; k++) {
```

```
        cout << match_box[P[i]].next[k] << " ";
    }
    cout << ">" << endl;
    cout << " result=" << match_box[P[i]].result << " ncase=" ;
    cout << match_box[P[i]].ncase << endl;
}
getchar();
*/

int lc, i;
char buf[128];
int type;

fprintf(stderr, " Key in type : 0:purposely 1:random 2:match_box 3:strictly 4:average ");
gets_s(buf);
type = atoi(buf);

lcount = 0;
while (1) {
    if (type != 4) {
        fprintf(stderr, " Learning mode .....................\n");
        fprintf(stderr, " Key in learning cycle : "); gets_s(buf);
        lc = atoi(buf);
        for (i = 0; i < lc; i++) {
            int len = execute(type);
            // print_path(Learn, i + 1, len);
            learn(len, type);
        }
        lcount += lc;
    }

    goodcount = 0;
    fprintf(stderr, " Test mode .....................\n");
    fprintf(stderr, " Key in Test cycle : "); gets_s(buf);
    lc = atoi(buf);
    for (i = 0; i < lc; i++) {
        int len = execute2(type);
        // print_path(Test, i + 1, len);
        int result = match_box[select[len - 1]].result;
        if (result == 1 || result == 0) goodcount++;
    }
    print_result(lc, type);
```

```
        /*
        for (int i = 0; i < 16; i++) {
            cout << i << " : " << "[ ";
            for (int k = 0; k < 9; k++) {
                cout << match_box[i].ban[k] << " ";
            }
            cout << "] <";
            for (int k = 0; k < 9; k++) {
                cout << match_box[i].next[k] << " ";
            }
            cout << ">" << endl;
            cout << "teban=" << match_box[i].teban  << " beans[] : [";
            for (int k = 0; k < 9; k++) {
                cout << match_box[i].beans[k] << " ";
            }
            cout << "]" << endl;
        }
        */

        fprintf(stderr, " Continue ? (CR for Yes) "); gets_s(buf);
        if (buf[0] != NULL) break;
    }

    return 0;
}
```

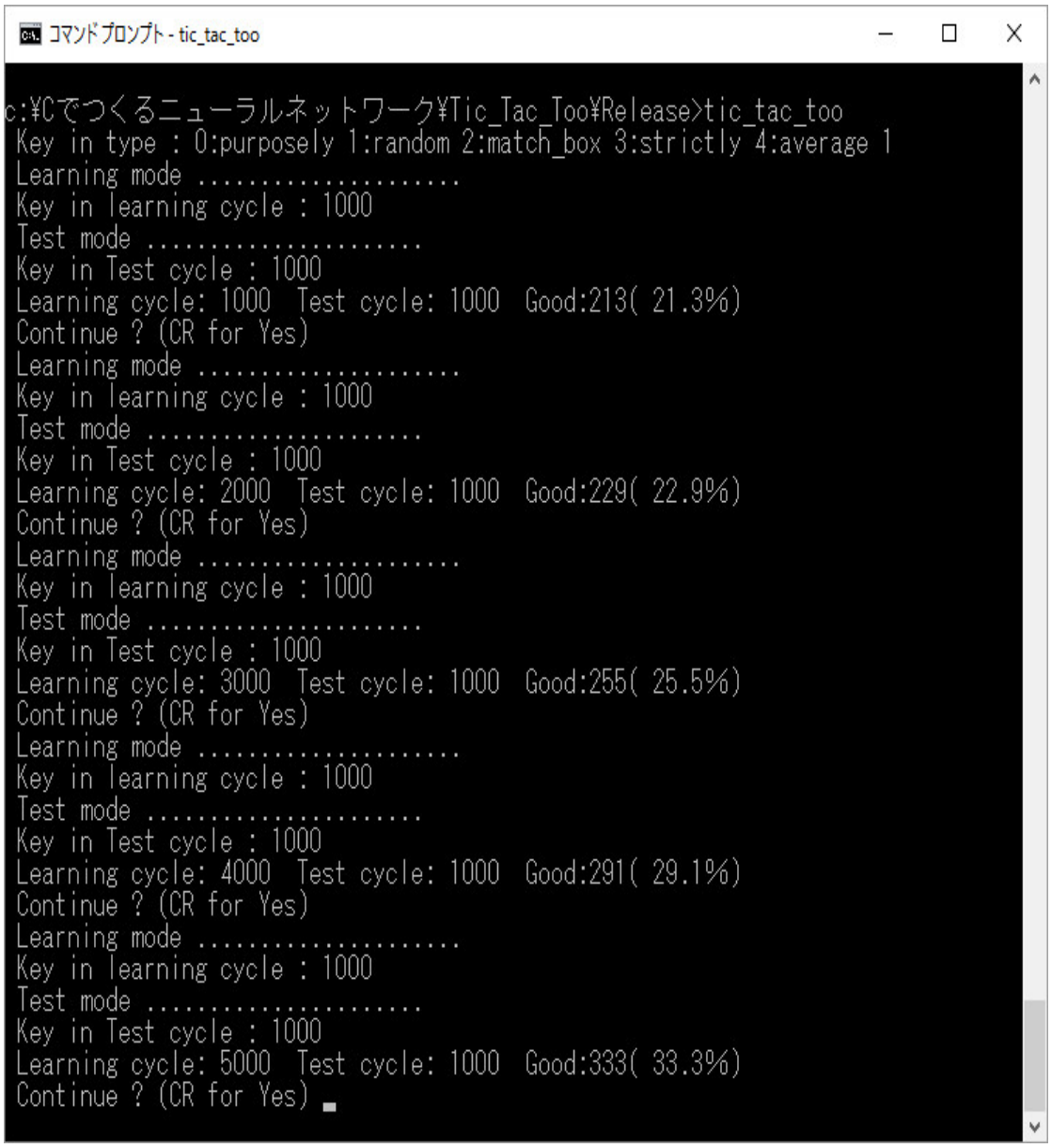
です。最初のプログラムの main() 関数を削除し、この部分を最初のプログラムに続けて書き込みます。

実行してみます。



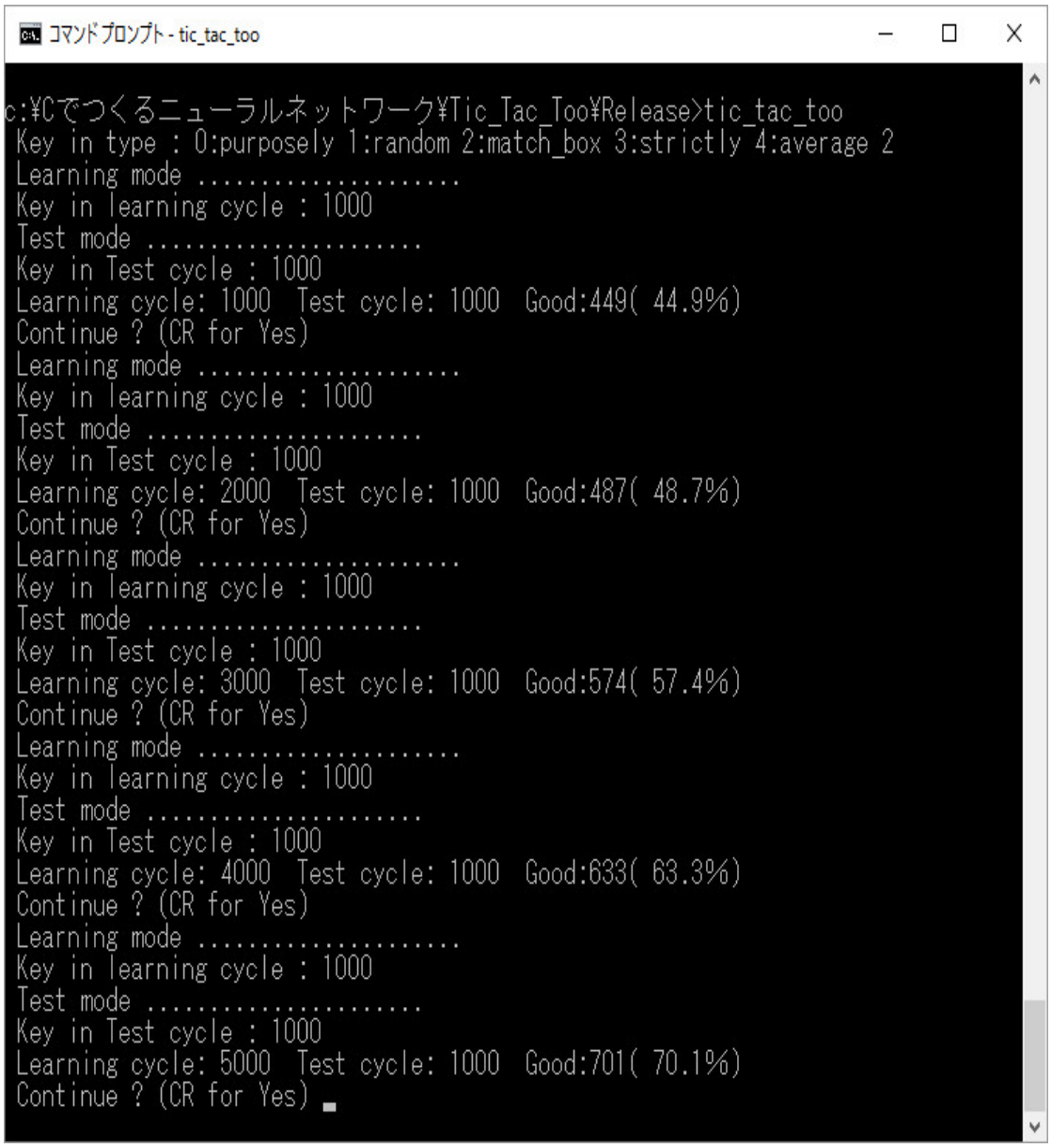
先手も後手もランダムに手を選ぶと先手が負けない確率は 73.9% です。但し、対称性を除いているので、正解の中央を選ぶ確率が９分の１ではなく、３分の１になっているので負けない確率は高くなっています。

　後手がランダムに手を選んだとき、先手の学習効果を見てみましょう。



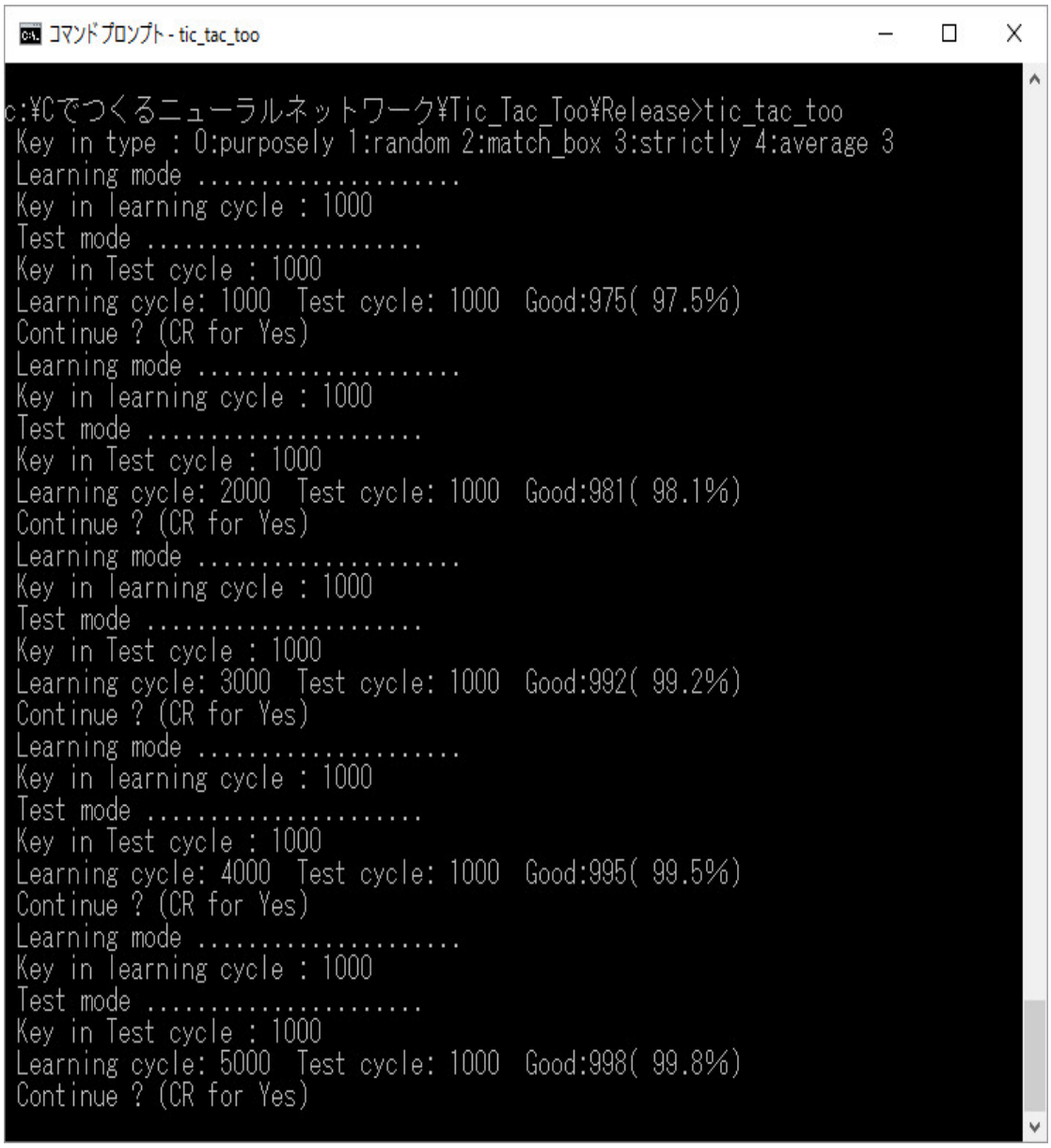
です。1000 回学習するたびに、21.3%，22.9%，25.5%，29.1%，33.3% と負けない確率が高くなっています。学習効果を見る場合、後手は最善手を常にさしています。

　後手もマッチ箱で学習して手を選んだとき、先手の学習効果を見てみましょう。



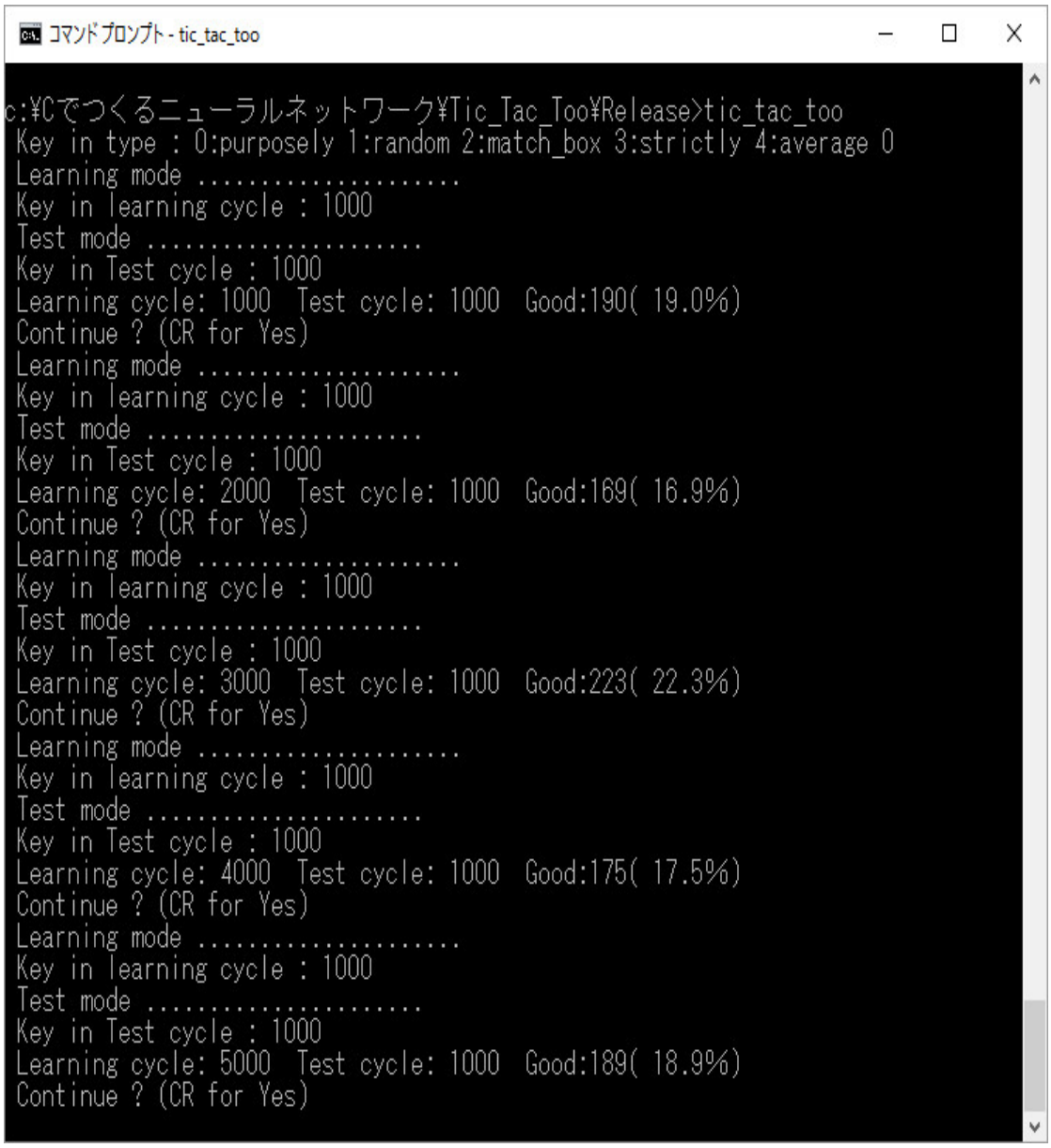
です。1000 回学習するたびに、44.9%，48.7%，57.4%，63.3%，70.1% と負けない確率が高くなっています。学習効果を見る場合、後手は最善手を常にさしています。後手がランダムに手を選んだとき、先手の学習効果を後手がランダムに手を選ぶものとし、後手もマッチ箱で学習して手を選んだとき、先手の学習効果を後手もマッチ箱で学習して手を選ぶものとして、つまり、それぞれ学習した相手との比較で計算すると、最初は後手がランダムに手を選んだときの方が成績は良いですが、３０００回学習すると後手もマッチ箱で学習して手を選ぶ方が出鱈目に手を選ぶ相手の場合の確率を超えました。判定基準を同じにしておかないと間違った結論を出してしまいます。

次にに後手が最善手を常に選ぶ場合の先手の学習効果を見てみましょう。



です。1000 回学習するたびに、97.5%，98.1%，99.2%，99.5%，99.8% と負けない確率が高くなっています。最初から高い確率で負けません。学習効果を見る場合、後手は最善手を常にさしています。

最後に後手が最悪手を常に選ぶ場合（常に可能なら常に負けてやる場合）の先手の学習効果を見てみましょう。



です。1000 回学習するたびに、19.0%，16.9%，22.3%，17.5%，18.9% と負けない確率は増加しているとは言えません。

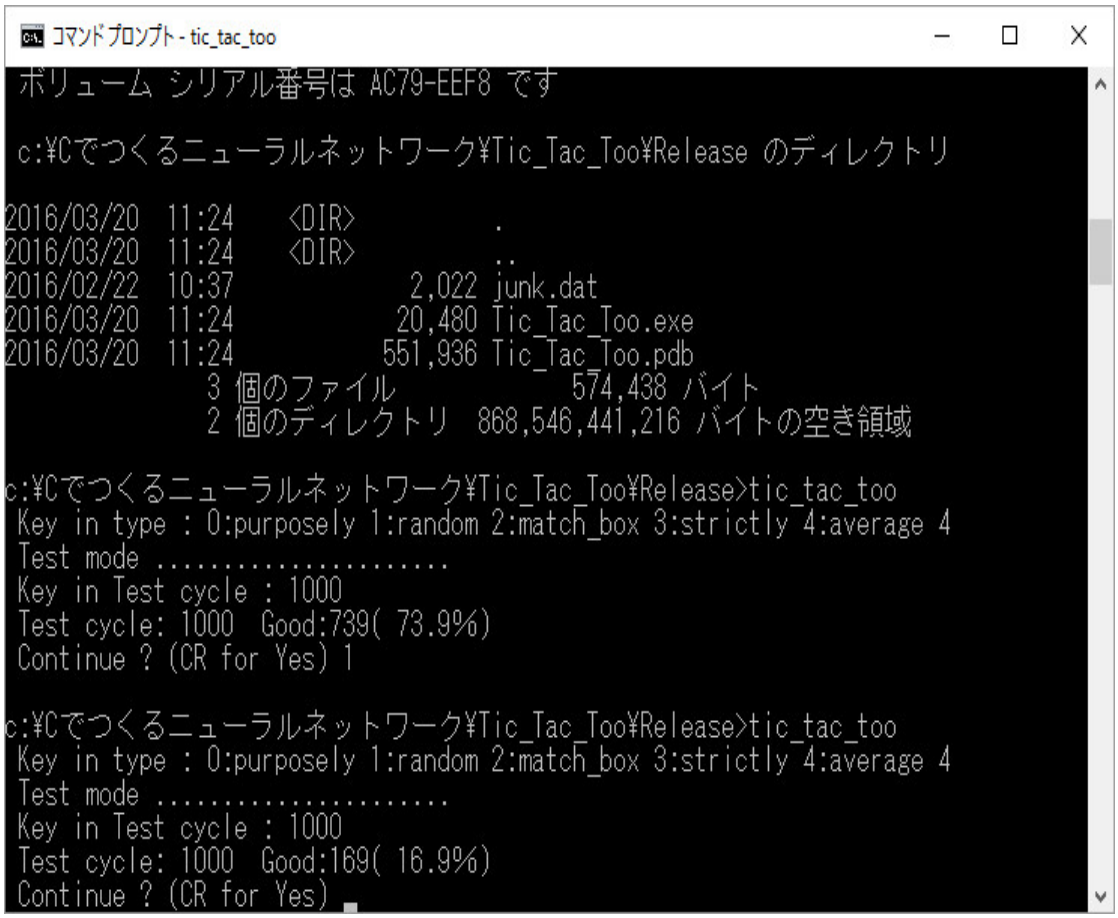
```
int execute2(int type) {
    int lv, ns;

    select[0] = nsselect[0] = 0;
    for (lv = 1; lv <= MaxLevel; lv++) {
        if (type != 4) {
            ns = next_select(select[lv - 1], 3);
        }
        else {
```

```
            ns = next_select(select[lv - 1], 4);
        }
        if (match_box[select[lv - 1]].beans[ns] - 1 < 0) {
            printf(" Error\n"); break;
        }
        select[lv] = match_box[select[lv - 1]].next[ns];
        nsselect[lv] = ns;
        if (match_box[select[lv]].ncase == 0) {
            break;
        }
    }
    return lv + 1;
}
```

を

```
int execute2(int type) {
    int lv, ns;

    select[0] = nsselect[0] = 0;
    for (lv = 1; lv <= MaxLevel; lv++) {
        if (type != 4) {
            ns = next_select(select[lv - 1], 3);
        }
        else {
            ns = next_select(select[lv - 1], 3);
        }
        if (match_box[select[lv - 1]].beans[ns] - 1 < 0) {
            printf(" Error\n"); break;
        }
        select[lv] = match_box[select[lv - 1]].next[ns];
        nsselect[lv] = ns;
        if (match_box[select[lv]].ncase == 0) {
            break;
        }
    }
    return lv + 1;
}
```

と修正して実行してみます。



先手はランダムに手をえらび、後手は最善手を常に選ぶと先手が負けない確率は 16.9% です。間違ったことを学習させると、全然効果がないとは言えませんが、学習を続けても効果は期待できません。学習効果を見る場合、後手は最善手を常にさしています。

これから指導者が適切でないと生徒の学習の伸びが遅いことが見て取れます。ただし、人間の子供を指導する場合は、わざと負けてやり、褒めて、自信を持たせることも必要で一筋縄ではいきません。良い指導者に恵まれることは難しいですが、良い本は自分で見つけることが出来ます。良い本に巡り合えれば、私のように独学でコンピュータを学ぶことが出来ます。ニューラルネットワークは熊沢逸夫著「学習とニューラルネットワーク」森北出版、小高知宏著「初めての機械学習」及び「機械学習と深層学習」オーム社のような C のプログラムがついている書籍から学び始めるのが私は分かりやすかったです。その後、気に入った他の深層学習の参考書を読めばいいです。64 ビットの Windows10 では、MinGW-w64 をインストールすれば、gcc 、 g++、 gfortran が使え、これらの本の C のプログラムが実行できます。そのとき、唯本を読んで、与えられたプログラムリストを学習するだけでなく、このように関連するプログラムを自力で作ってみることが大事です。他人が作ったプログラムリストを研究するだけでは、失敗しないのであまり勉強になりません。成功体験より失敗したことの方がだいじです。あらゆる失敗を経験したから成功したとよく聞きます。上のプログラムリストにあるコメントアウトした部分は、プログラムのバグを調べるために、コンピュータが実行したことを確かめるためのものです。普通、プログラムが出来たと思ってから実際にプログラムが出来るまで倍以上の時間がかかります。自分でプログラムを作ると必ずと言っていいほど間違っています。思い違いしている所、間違っている所をプログラムを動かし、その動き

を探索し、バグを探し、それを修正しを繰り返します。このデバッグをすることでプログラミング技術が上達します。多くの人がこのデバッグの作業に耐えられずにプログラミングに挫折します。こんな簡単なプログラムでも、本を読んで、サンプルプログラムを打ち込んで、それを真似して作るのに丸２日かかりました。時間をかければ長い目で見れば、やはりそれだけの報酬があります。有名な「マシュマロ・テスト」のように我慢した人にはより多くの報酬が約束されます。