VisualStudio 便利機能集 Ver. 1.10

最終更新日　2011/6/25

せっかくの統合開発環境なのだから、いろいろ機能を使ってデバッグ効率を上げようってことで軽くまとめてみました。

使用環境

VisualC++ 2008 Express Edition 及び一部 VisualStudio 2008 Professional
（多分 2010 や Professional Edition でも同じ）

更新履歴

ver 1.10 分け方を大幅変更。加筆修正などなど。
ver 1.00 とりあえず一旦完成ってことで 1.00 に。
ver 0.10 修正と微調整。
ver 0.08 ５節 5-5 追加。おまけでショートカットキーまとめ（最後のページ）
ver 0.06 5-4 クリップボードリングの説明追加
ver 0.05 ５節に３つ追加。ちょっと修正
ver 0.02 微調整、加筆。５節の追加
ver 0.01 とりあえず作ってみた

※後から気がついたものは全部５節につっこむ。

目次

0. ショートカットキー関連

0-1.他の主要なアプリケーションでも共通するもの

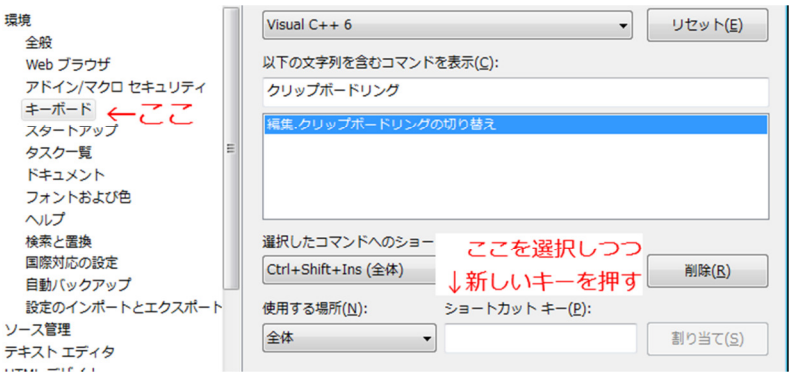
Ctrl + C → コピー
Ctrl + X → 切り取り
Ctrl + V → 貼り付け
Ctrl + Z → 元に戻す(アンドゥ)
Ctrl + Y → やり直し(リドゥ)
Ctrl + S → 保存
Ctrl + A → 全範囲選択
Alt + F4 → ウインドウを閉じる
Alt + Tab→ アクティブウインドウ切り替え

0-2.VisualStudio ショートカットキー（1 節以降で扱わないもの）

Ctrl + F5 → デバッグなしで実行
F5 → デバッグ実行
F7 → ビルド
Ctrl + Alt + F7 → リビルド
Ctrl + Break → ビルドの中止(Break キーが無かったら pause キーで)
Ctrl + Shift + U→ 範囲内の文字をすべて大文字にする。
Ctrl+] →（括弧部分にカーソルを合わせて）対応するかっこへ飛ぶ

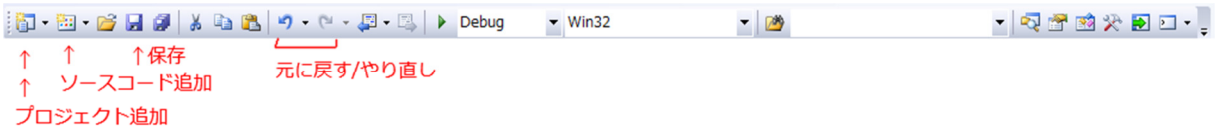
0-3.ショートカットキーの変更

使いやすいショートカットキーによく使う機能を設定しておくとコードを書いたり、デバッグしたりする効率が上がります。
やり方
「ツール」→「オプション」
（2-1 行番号の表示を参照）
「環境」→「キーボード」
あとは必要なコマンドを検索して自分の好きなショートカットキーに設定してください。



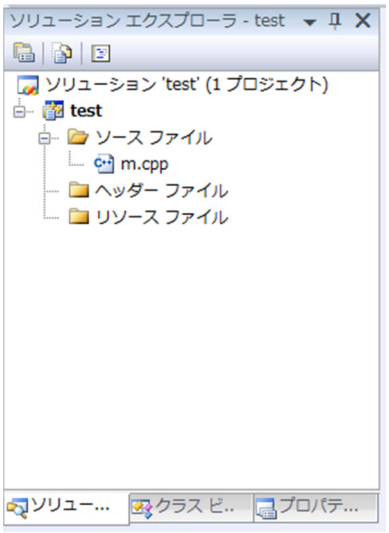
1 .VisualStudio の基本

1-1.標準ツールバー



上記のアイコンくらいは知っておくと便利です。

1-2.ソリューションエクスプローラ



右図。
右クリックでのソースファイルやヘッダーファイルの追加、プロジェクトのプロパティの設定、ファイルが増えてきたときの切り替えなどによく使う。
ウインドウが出ていなかったら、「表示」→「ソリューションエクスプローラ」または Ctrl+Alt+L で。

1-3.意外と知らない単語と意味

・ソリューションとプロジェクト
プロジェクトが EXE ファイルや DLL ファイルを１つ作成するのに必要なもの。
ソリューションは複数のプロジェクトをまとめて管理するフォルダのようなもの。
とはいえ、小規模ではソリューション内に１つのプロジェクトしか含まないので、
ほぼソリューション≒プロジェクトとみても問題なさそうです。

・ビルドとコンパイル
コンパイルはソースコード(.c / .cpp)を解析し、オブジェクトファイル（.obj）を作成する。
ビルドはコンパイルに加え、各オブジェクトファイルをリンクし、実行可能ファイル(.exe)を作成すること。

1-4.デフォルトの各種ファイルの場所

プロジェクトの保存設定を変えていなければ、プロジェクトのデフォルトの保存場所は、（マイ）ドキュメント→Visual Studio 2008→Projects の中にあります。

階層は

プロジェクト名（フォルダ）

Debug　→中に実行ファイル(exe)を含む

(Release) →リリースビルドで作られる。exe を含む

プロジェクト名.sln（ソリューションファイル）

プロジェクト名.ncb（Intellisense 用データベース）

プロジェクト名.suo

プロジェクト名(フォルダ)※

ソースコード

プロジェクト名.vcproj(プロジェクトファイル)

Debug →obj ファイルとか

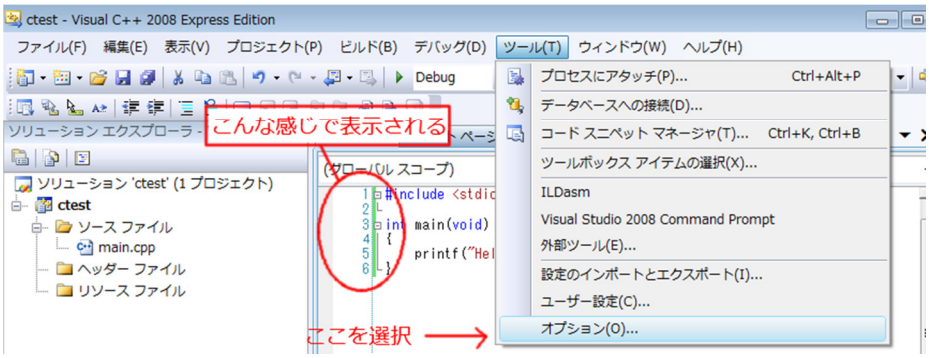
※素材を使う場合、VisualStudio で開発している間は赤字で書いたプロジェクト名(ソースコードのあるフォルダ)がカレントディレクトリ（基準となるフォルダ）になるので、このフォルダに素材を入れる。

2.コードと見た目

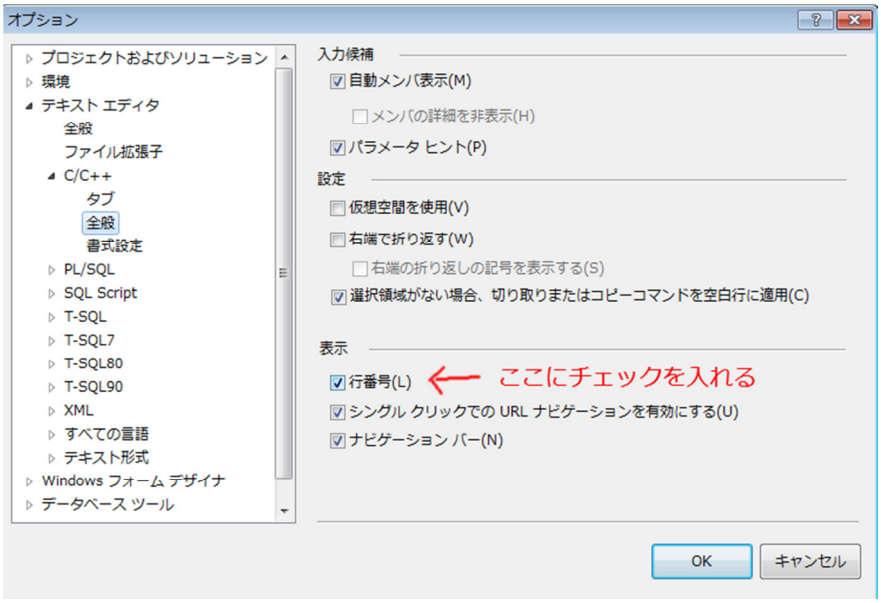
2-1.行番号の表示

ソースコードの左側に行番号を表示する。（デフォルトでは OFF）
デバッグ情報に何行目とか出てくる場合が多いので設定しておくと役に立つかも。
単純に何行くらい書いたか知りたいときにも。

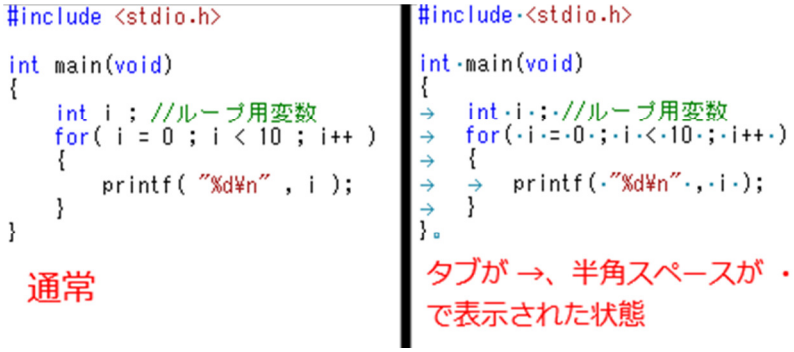
設定方法：
「ツール」→「オプション」を選択。



以下のようなウインドウが開くので、「テキストエディタ」→「C/C++」→「全般」を選び、行番号のチェックボックスにチェックを入れる。



2-2.タブと空白の視覚化及び解除方法 ～ Ctrl+R+W



設定方法：

「編集」→「詳細」→「スペースの表示」を選択する。

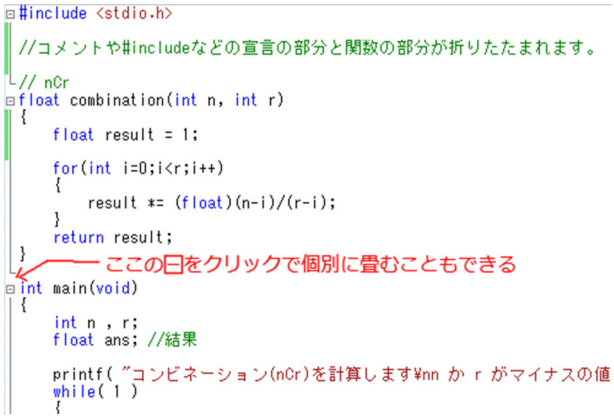
または、Ctrl+R+W を押す。

2-3.コードの折り畳み ～ Ctrl+M+L

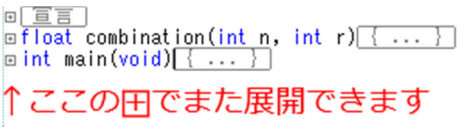
長くなった関数やコメントアウトなどの部分を折りたたむ機能。

開発規模が大きいほど便利かと思われます。

こんなコードが↓



全部折りたたむとこのように。



設定方法：

左の－/＋で折りたたむ/展開。折りたたみ状態なら｛...｝をダブルクリックしても展開できる。

Ctrl +M+L でそのファイル内を一括して折りたたみ/展開できる。

2-4.インデントの自動調整 ～ (Ctrl+A →)Ctrl+K+F

コードを見やすくするインデント（字下げ）。

いつの間にかインデントがめちゃめちゃになっていたり、そもそも考えていなかったりした時の簡単な設定方法。

↓こんな感じに自動で字下げしてくれます。

before

```
#include <stdio.h>
float combination(int n, int r)
{
float result = 1;
for(int i=0;i<r;i++)
{
result *= (float)(n-i)/(r-i);
}
return result;
}
int main(void)
{
int n , r;
float ans; //結果
printf( "コンビネーション(nCr)を計算します¥nn かr が
while( 1 )
{
printf( "n->" );
scanf( "%d" , &n );
printf( "r->" );
scanf( "%d" , &r );
if( n < 0 || r < 0 )
break;
ans = combination( n , r );
printf( "combination( %d , %d ) = %f¥n¥n",n,r,ans );
}
}
```

after

```
#include <stdio.h>
float combination(int n, int r)
{
    float result = 1;
    for(int i=0;i<r;i++)
    {
        result *= (float)(n-i)/(r-i);
    }
    return result;
}
int main(void)
{
    int n , r;
    float ans; //結果
    printf( "コンビネーション(nCr)を計算します¥nn
    while( 1 )
    {
        printf( "n->" );
        scanf( "%d" , &n );
        printf( "r->" );
        scanf( "%d" , &r );
        if( n < 0 || r < 0 )
            break;
        ans = combination( n , r );
        printf( "combination( %d , %d ) = %f¥n¥n",
    }
}
```

設定方法：

Ctrl + K + F で”選択範囲内”の自動字下げを行います。

ファイル全体に自動インデントをかけたい場合、Ctrl + A(全範囲選択)を先に行い、全体を選択状態にしてから Ctrl + K +F を押すと全体にかけられます。

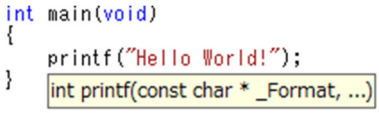
3.入力補助

3-1.Intellisense（入力候補表示及び補完、オートコンプリート/オートコレクト）

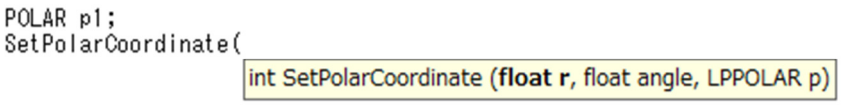
インテリセンス（Intellisense、VisualStudio にある入力支援機能）を利用した入力補完。
入力候補の表示（オートコンプリート）とタイプミスの補正機能（オートコレクト）を持つ。
うまく使えば非常に便利。

関数の引数リストなどの情報は関数名にカーソルを合わせたり関数の引数を入れ始めたりした際に表示される。
もちろん自作関数にも対応。

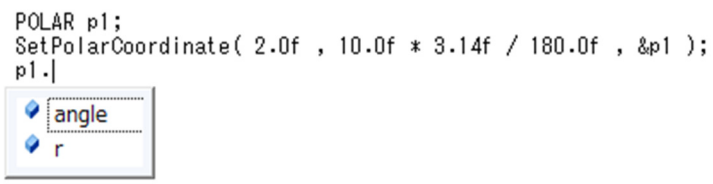
（例 1）こんな風に引数リストを表示したり（printf は文字列(char *)を取ることがわかる）



（例 2）
自作関数や構造体でも問題なし。



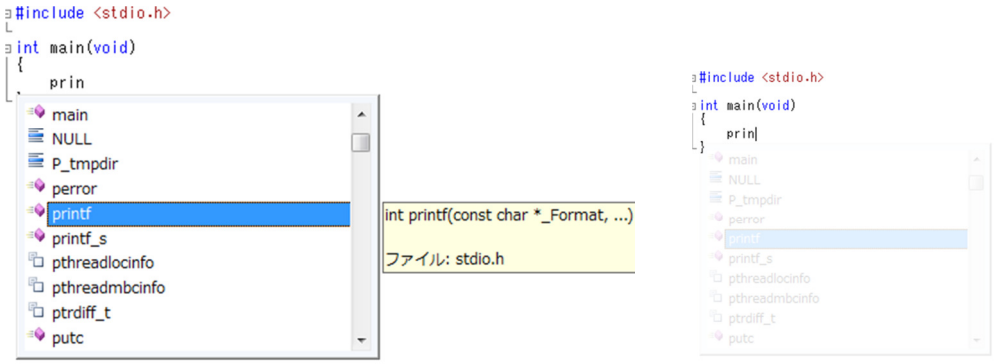
構造体やクラスの場合はメンバ変数やメンバ関数も表示してくれる。



Ctrl + Space →オートコンプリート/オートコレクト

途中まで関数名を打ったりしてから Ctrl + Space を押すと入力候補を表示してくれます。

↓こんな感じで入力候補を表示してくれる



（後ろのコードが見たかったら Ctrl を押すと薄くなります。↑）

※入力候補が一つしか考えられないような関数の場合、Ctrl + Space を押した時点でその関数名に置き換わります。

※Intellisense が機能しなくなった時の対処法
いろいろクラスや関数を追加したり消したりとコードが長くなってくると稀に Intellisense が更新されなくなることがあります。
そういう場合は ncb ファイル(1-4.デフォルトのファイルの場所を参照)を削除して Intellisense データベースを再構築させると復活するかもしれません。

また、単純にソースコードが間違っている場合も多々あるので更新されない場合はまずソースコードを見直してみるべきかも。

3-2.矩形選択 ～Alt 押しながら選択

Alt を押しながら選択すると矩形（長方形）の形で選択できます。

```
// nCr
float combination(int n, int r)
{
    float result = 1;

    for(int i=0;i<r;i++)
    {
        result *= (float)(n-i)/(r-i);
    }
    return result;
}
```

3-3.クリップボードリング ～ Ctrl+Shift+Insert

VisualStudio ではコピーや切り取りでクリップボードに保存された情報は一定個数保存されます。

普通に貼り付けるなら Ctrl+V ですが、Ctrl+Shift+Insert でクリップボード内に保存された情報を順々に取り出して貼り付けることができます。

（編集→クリップボードリングの切り替え でも可）

3-4.コメント ～ Ctrl+K+C

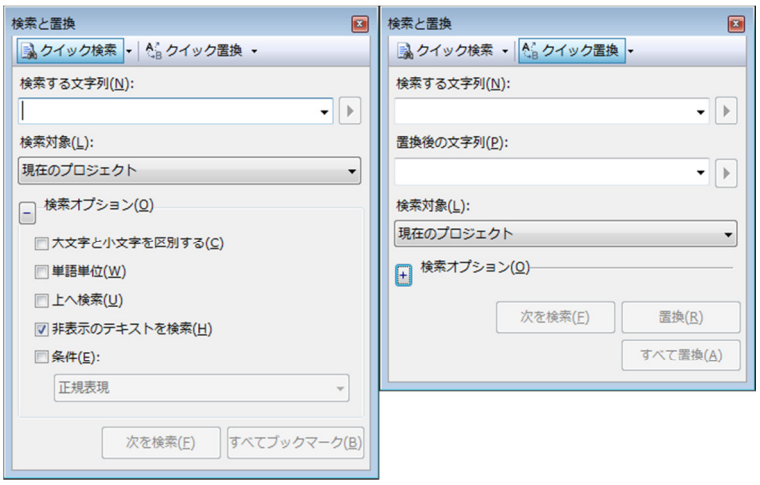
Ctrl + K + C で選択した範囲またはその行にコメントを挿入。

また、Ctrl + K + U でコメントを解除できる。

4.コードの検索

4-1.検索と置換 ～ Ctrl+F 、Ctrl+H、及び Ctrl+F3

VisualStudio に限った機能ではないのですが、知っておくと便利なので一応。
検索は Ctrl + F , 置換は Ctrl + H でできます。
また、Ctrl+F3 で現在カーソルが合っている単語をそのまま検索できます。



4-2.VisualStudio 内でのウインドウ切り替え ～ Ctrl+Tab

Alt + Tab の VisualStudio の内部だけのバージョンといった感じ。
Ctrl + Tab で VisualStudio 内でのアクティブなタブの変更ができます。
Ctrl を押しながら矢印キーまたはマウスで選択。

↓こんな感じのがでてくる

4-3.宣言部へジャンプ　～ F12

F12 キーを押すと選択した変数や関数の定義部分にジャンプできます。

```
float combination(int n, int r)
{
    float result = 1;

    for(int i=0;i<r;i++)
    {
        result *= (float)(n-i)/(r-i);
    }
    return result; ←この変数の元を探したい
}
```

→

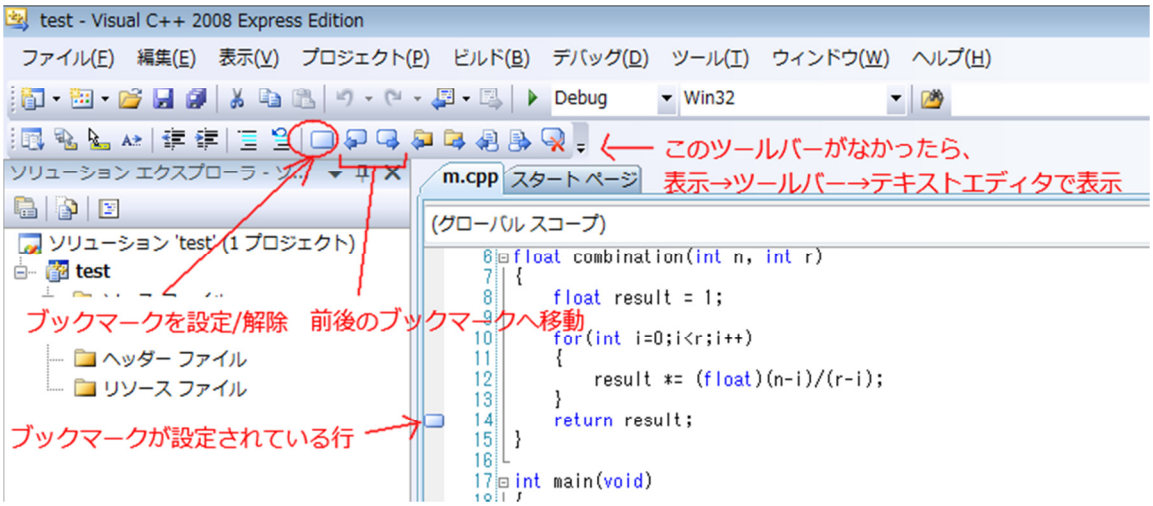
```
float combination(int n, int r)
{
    float result = 1; ←宣言部分にジャンプ

    for(int i=0;i<r;i++)
    {
        result *= (float)(n-i)/(r-i);
    }
    return result;
}
```

4-4.ブックマーク

コード中の特定の行にブックマークを設定しておくと、その場所へジャンプできる機能。
コード中の特定箇所を行き来したりするときに便利。

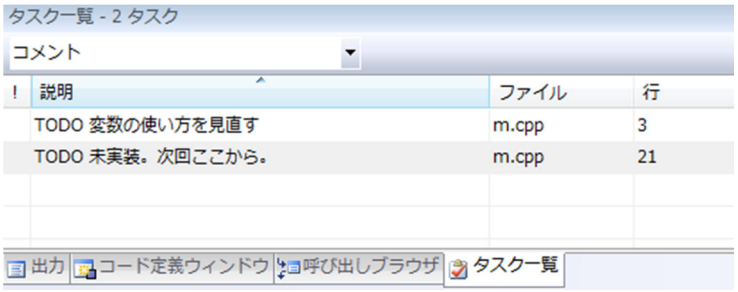
Ctrl + K + K でブックマークの設定
Ctrl + K + N(または F2) で次のブックマークへジャンプ
Ctrl + K + P(または Shift+F2) で前のブックマークへジャンプ
↓ツールバーのアイコンでも使えます。



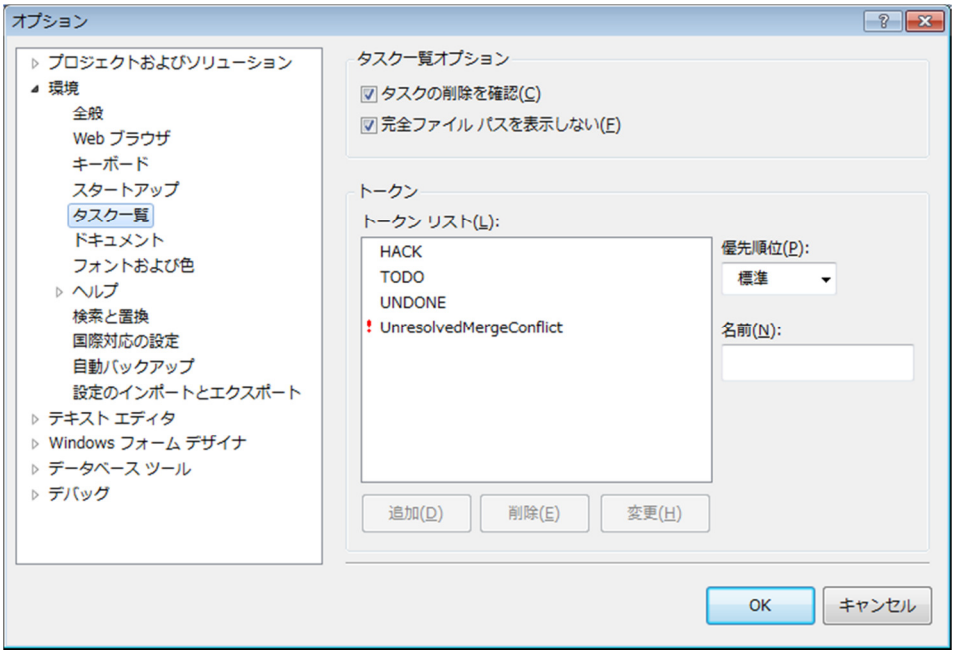
4-5.タスクー覧ウインドウ

ソースコード中に特定の形式でコメントを書いておくと、それを見つけて表示してくれるウインドウ。形式は //TODO など。

タスクをダブルクリックするとその行に飛べます。



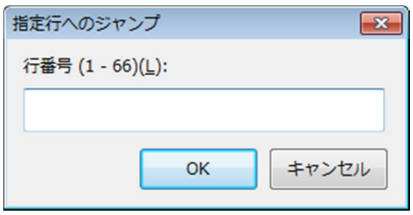
表示する形式は「ツール」→「オプション」→「環境」→「タスクー覧」で見たり追加、削除したりできます。



4-6.指定行へジャンプ ～ Ctrl + G

Ctrl+G で指定行へジャンプできます。

エラーメッセージで特定の行にあることがわかった場合などに。

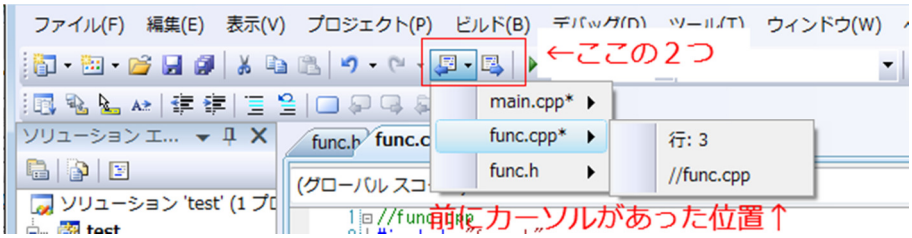


4-7.ナビゲーション機能による前カーソル位置への移動

以前カーソルが合った位置に戻ったり、進んだりする機能。

テキストツールバー上にボタンがある。

検索などでなんども移動した後にもとの場所に戻りたい場合などに。



5.ブレークポイント関連

5-1.ブレークポイント（F9）

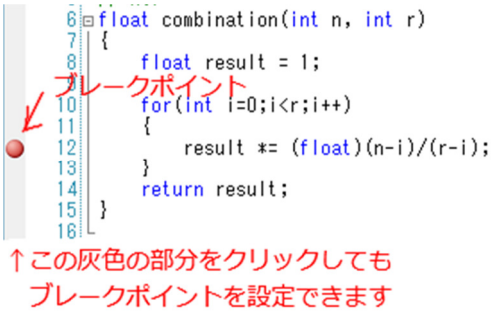
プログラムの動作を途中で止めることのできる機能。

特定の行に設定し、プログラムの処理がその行に来た時にそこで一旦プログラムを停止させる。

VisualStudio でのデバッグの要。超重要。

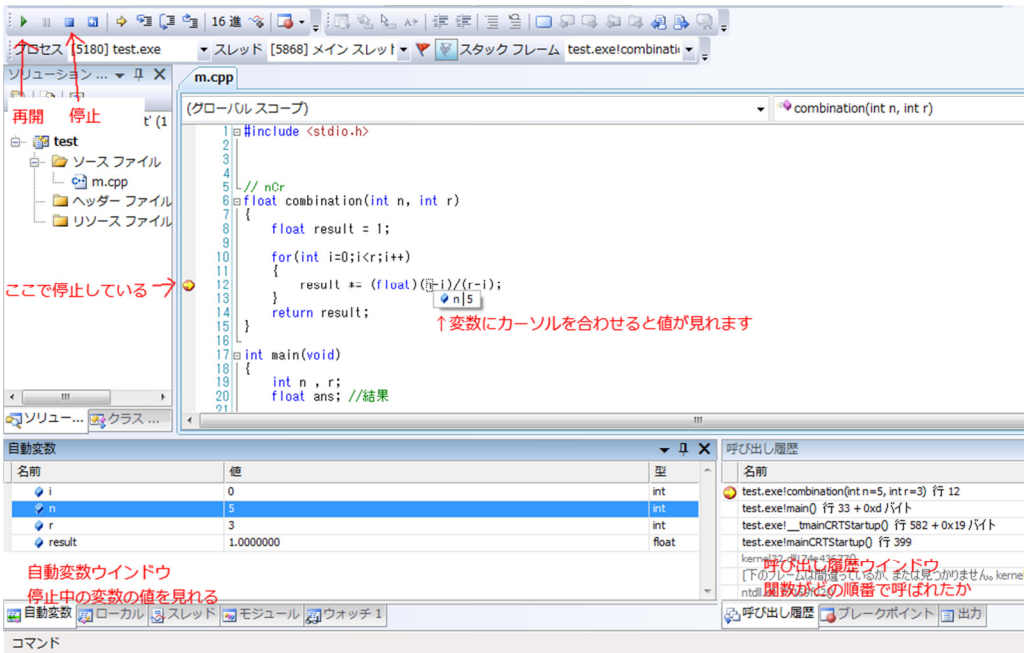
デバッグ実行（F5）時のみ有効。

F9 キーで設定可能。



４の項で扱う各種サブウインドウと組み合わせると非常に強力。

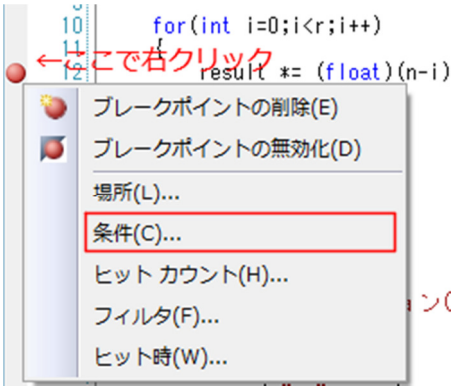
下のように動作します。



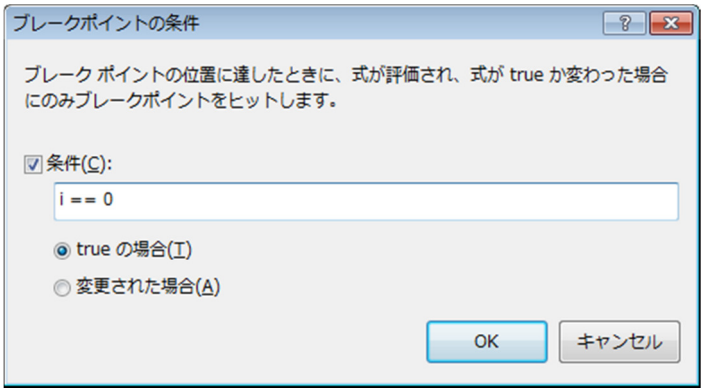
5-2.条件付きブレークポイント

特定の場合にしか止まらないというようなブレークポイントが作成できます。

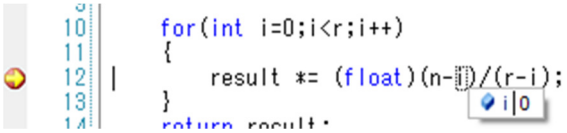
条件をつけたいブレークポイント上で右クリック



あとは条件を設定するだけ。その行で使える変数であればローカル変数を条件に入れても大丈夫です。

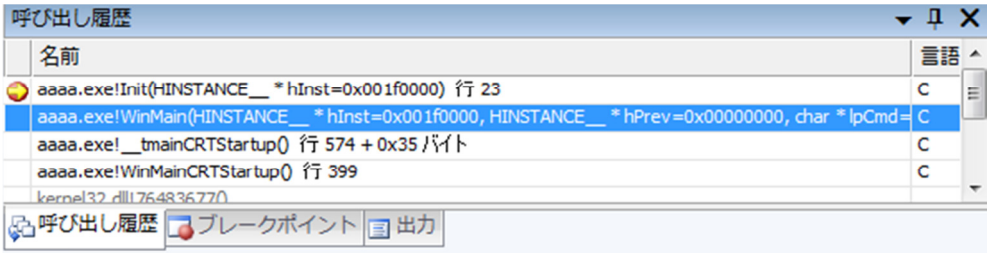


実行結果↓ i == 0 のときだけブレーク



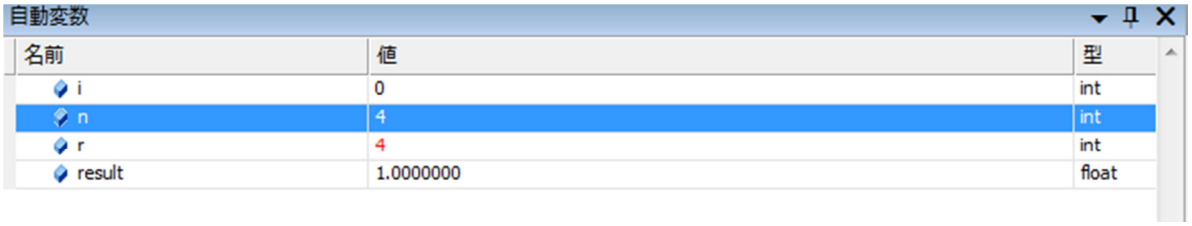
5-3.呼び出し履歴

ブレークポイントで止まった時に、その時点までで関数がどう呼ばれたかを見ることのできるウインドウ。



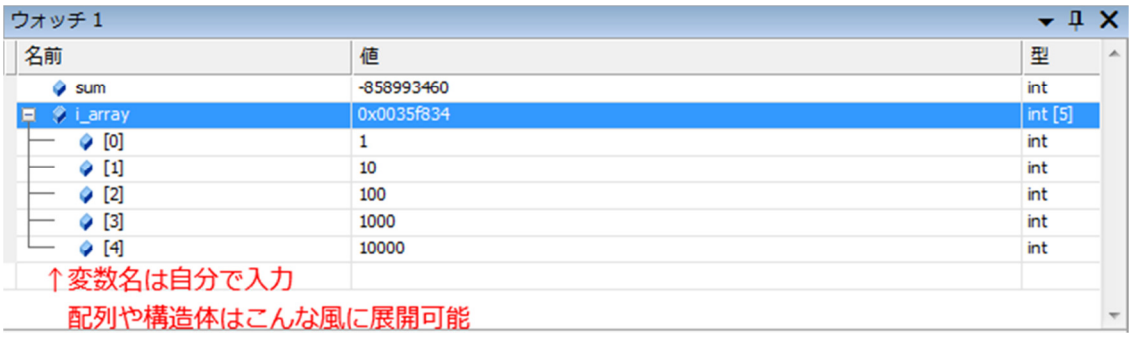
5-4.自動変数、ローカル

ブレークポイントで止まった時に、使われている変数の値を見ることのできるウインドウ。
自動変数はその時点（その行と前の行）で使われているもの。
ローカルはその時点でのローカル変数
ウォッチは自分で変数名を入力し、その変数があったらその値を調べられる。
↓自動変数

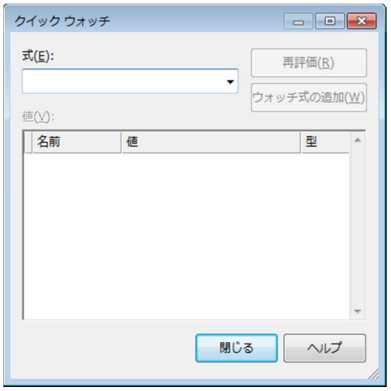


5-5.ウォッチ

ブレークポイントで止まった時に、使われている変数の値を見ることのできるウインドウのひとつ。
普段は何も表示されてないが、自分で変数名を入力すると、その変数があったらその値を調べられる。
↓ウォッチウインドウ



クイックウォッチなんてものもある（右の画像）
ショートカットキーは Shift + F9 または Ctrl + Alt + Q

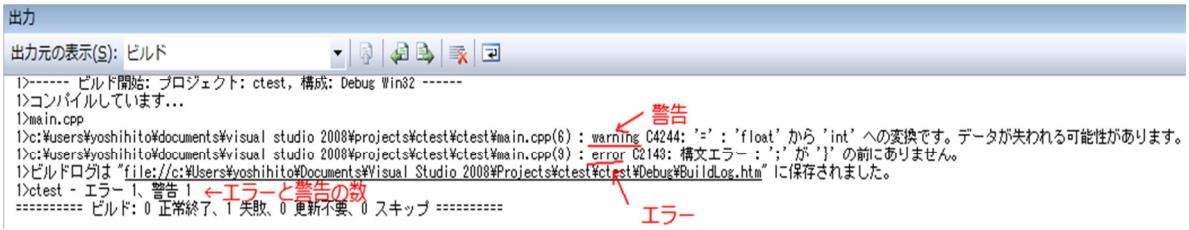


6.いろいろなサブウインドウ

6-1.出力

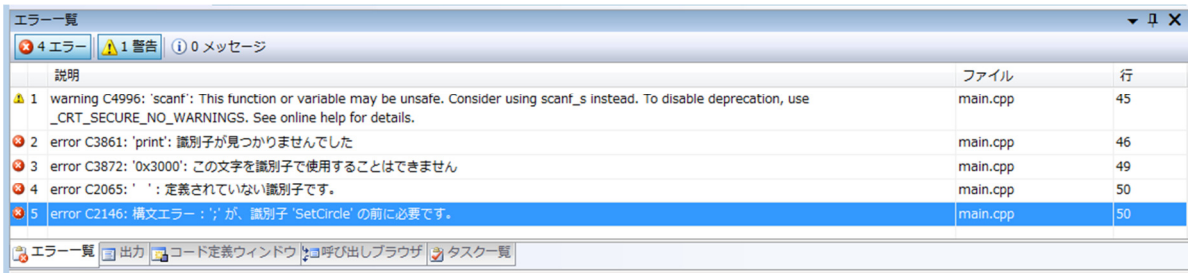
エラーや警告が表示されるウインドウ。

出ていなければ「表示」→「出力」から。



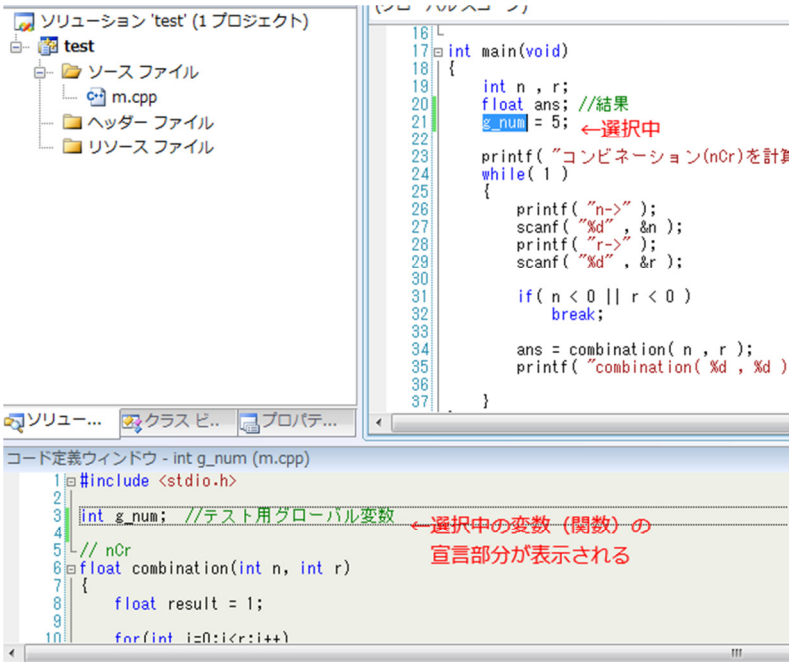
6-2.エラー一覧

エラーと警告を表示するウインドウ。



6-3.コード定義

選択した変数や関数の定義部分を探して表示する。読み取り専用ウインドウ。



7.その他

7-1.VisualStudio で使える#pragma 指令

コンパイラ固有の指定を行う構文。ソース中に書く。
Visual Studio では主に以下のものが使えます。
開発中にコンパイラが変わったりする場合は使わない方が無難。逆に一つのコンパイラで開発するならどんどん使って大丈夫です。

①#pragma once
ヘッダーファイルの先頭に書くとそのヘッダーファイルが２回以上インクルードされなくなる。（インクルードガード）

②#pragma warning (指定:警告番号)
指定は主に以下の２つ
disable →対象の番号をもつ警告を表示しない
error　→警告をエラーに格上げ
push とか pop というのもあるそうです。詳しくは調べてください。

警告番号には C〇〇〇〇の〇〇〇〇の部分を入れる。
（例）警告を消す。
warning C4996: 'scanf': This function or variable may be unsafe. （略
↓これを消すには
#pragma warning (disable：4996)
と書く。
基本的にあまり多用してはいけないとは思いますが、便利な場合もあるので。

（例）逆に警告をエラーに引き上げる場合
warning C4700: 初期化されていないローカル変数'i' が使用されます
#pragma warning (error：4700)
↓これを書くと
error C4700: 初期化されていないローカル変数'i' が使用されます
とエラーになる。
未初期化はエラー扱いの方がいいと思いますがなんでデフォルトが警告なのでしょうか

③#pragma comment (lib , "ライブラリ名")
リンクするライブラリの埋め込み
あまり使ったことないので詳細は割愛。

④#pragma message("表示したいメッセージ")
出力ウインドウへの簡単な文字列出力。

例）:
#pragma message ("このメッセージが出力されます")
↓出力ウインドウに表示される。
1>このメッセージが出力されます