ゲームアーキテクチャ

はい、ゲームアーキテクチャというヨクワカラナイ授業が今年から追加されました。
担当は、昨年まで常勤やってたけど、今年から非常勤になった川野です。

基本的に１年生からは隔離されていたので、初めて目にする人も多いと思いますので、自己紹介ししとこう。

名前…川野 竜一(Kawano Ryuichi)
前々職…大阪の某ゲーム会社で KOF とかの格闘ゲームを作っていました
教える内容:だいたい数学とか、ゲームアルゴリズムとか、CG のアルゴリズムとか周り
現在…フリーランスで非常勤なのに週 10 コマも担当しつつ、レンダリングエンジンの開発のお仕事とかやっています
趣味…ボクシングと自転車と猫と、CEDEC とかの公募に応募する事、あとなんかゲーム作りのコンテストとかに応募する事…とにかく目立ちたい
著書に『DirectX12 の魔導書』があります。あと、Cedil で僕の名前で検索するといくつか出てきます。何故か『まんがとイラストで分かる GPU 最適化』って本の後ろに僕の名前が載ってます。

な感じ。では始めましょう。

目次

はじめに

この授業が何のための授業かという事なんですが…まぁ、正直頼まれた内容がふわっとしすぎてて、僕としてもなんて言ったらいいのかよく分かりません。

昨年までは最上級コースの開発全部見てた感じだったんだけど、非常勤だから「**なんかクオリティアップにつなげられるような内容にしてください！**」という、本っ当にふわっとした要求。なめとんのか!!

しかも、まぁ、ぶっちゃけた話、学生さんのプログラミングレベルにかなり差があるという…これが一番つらいんだけど僕としてもいちいち別々に授業組む余裕もないので、ある程度は統一します。

進度と深度をちょっと変えるくらいかな。
基本的に僕はゲームのプログラムしか教えられんので、この時点で IT 系に行くとか、一般職に行くとか言う人に対応しません。
色んな学生さんがいるって事は知ってるので、全員が全員ゲームプログラマを目指しているわけじゃないのは分かってるけど、それはそれとして、課題はきっちり出してもらいます。

ともかく「みんながゲームプログラマを目指している」という体(てい)で授業しますし、お話します。そもそも目指してない人は何でここにいるの？って感じなんだけど、まぁ何かしらの事情があるんでしょう。とはいえ、高え学費と、1 日のうちの何時間も消費してる事は自覚して、きちんとそれだけの対価を支払ってるという認識を持ってください。

その結果、ここにいるべきかどうかを判断するのは自分です。もう二十歳くらいにはなってるんだから、その選択については自分で責任を持ってください。

「目指さないんだったら出ていけ」と言ってるわけじゃないですよ？全然目指してなくてもなんとか 3 年なり 4 年やり過ごしてただ卒業する意味があった事例もたくさんありますからね。

ただ、特に次年度就職年次の人は、少し覚悟をしてもらった方がいいと思います。生半可な覚悟だと精神的にきついです。今のウチに言っておきます。

何でこういう事言うのかというと、昨年までに悲しい、つらい状況を沢山見てきたからな

だよね。もし本当に目指す気があるんだったら、作る事を習慣づけたうえで、きちんと戦略を練ってください。

習慣について

何でここでいきなり習慣の話をするのかというと、**失敗する奴ってのは無理をしがち**だからです。たまに無理をするのならまだいいのですが、目標そのものが不可能な目標だったり、そもそも自分でも到達できると思ってなかったりしてて全くもって意味がない事になってる事が多いんですよね。

とりあえず次年度の人は今 4 月でしょ？で、就職活動がだいたい 12 月半ばくらいから始まります。つーわけで約 10 カ月。この間に就職活動ができるだけのゲームを作れるようになってる(そして実際に作ってる)必要があるわけです。

10 カ月というと約 300 日…土日を含めるか含めないかはお任せしますが、300 日まるまる使えるわけじゃないと考えると約 220 日くらい？の間にどこまで行けるかがカギなのですが
できない奴ほど無理をしようとします。

ちょっと言っておくと、できない奴ほど、ほんの少しずつほんの少しずつでいいからできるようになる事を考えてください。**いきなりは無理!!**
現実の世界に覚醒とか…ないから!!

授業で分からなかった部分を放課後に 30 分だけ復習するとか、そんなんでいいです。それすら難しいとは思いますが。ただ**この 30 分を怠ると課題提出前に何時間も、集中力もないまま苦痛な時間を過ごした挙句に何の成果もなく、能力も進歩しない**ってのはよくある事です。

無理をしないように、30 分がきついならまずは 10 分でも 5 分でもいいので、ちょっとだけ残って他の人が遊んでいる間に復習しましょう。何度も言いますがここで無理してはいけません。
最初はモチベーション高いんで 3 時間とかやっちゃいますがだいたいすぐ挫折します。
3 時間くらいが当たり前の人は良いですが、今現在できてない人にとっては 30 分すら大変なはずです。10 分から始めましょう。

よくいるのが3~4年ゲームプログラミングやってきたのに if 文や for 文も分からない人がいます。うん、全くの無駄なので、まずはそういう部分だけでも理解する所からやっていきましょう。

とにかく**無理をせず、「最初は習慣作り」と思って「なめとんのか？」ってくらいゆっくり始めましょう。**まだまだスタート地点です。エンジン全開まではまだ早い。

ゲームプログラマになりたきゃ、ほんの少しずつでいいからそこに近づく努力をすることを習慣化しましょう。意志の力だけでは…無理です。

そして、ごくごくたまになら無茶をしてもいいです。たまには無茶をしましょう。

さて、じゃあガムシャラに習慣化して努力してりゃいいのかというとそうでもない。アホは戦略も立てずにやろうとして徒労に終わる。

戦略について

戦略って何かっちゅーと、最終目標に到達するためのやり方って事ね。で、これは人によって違う。その人の特性によって違うから、自分で考えなきゃいかんわけ。

さっきも言ったけど、次年度なら残り 10 カ月。この 10 カ月の間に何を用意するのか…簡単に言うと計画なんだけど、計画を立てるだけでも夕メなんだよね。自分の持ち味をどう活かすのかを考えないと。

多分、皆さん少なくとも 1 年以上はプログラムしてるからもう薄々分かってると思うけど、同級生の中には「クッソ強いやつ」がいると思う。到底こいつには勝てないな…と。

こういう奴と張り合う必要は、ないです。

どうも就職活動を学校のテストと同じように思ってる人がいるかもしれんけど、ちょっと違う…いや、そんなに離れてるわけじゃないと思うけど違うのよね。何が違うのかというと、会社によって**そもそものニーズ**が違うし、自分の特性上としても伸ばせる部分とそうじゃない部分があるわけ。

まず戦略のポイントとしては

- 会社(大雑把でいい)選び
- 自分の何をウリにするのか

この 2 つがポイントなのよね。まぁ～だ実感がないかもしれないので、やれ任天堂だ、バンナムだ、サイゲームズだなんて出てくると思うんですが、そんなん就活考えてるとは言えないわけ。

あ、レベルが高すぎるって意味じゃないよ？確かにレベルが高いけどさ、こういう会社の名前が出て来るって事は正直その会社について…その会社の開発の仕事についてな〜んも分かってないって事。小学生が自分の希望を言ってるのと変わらないわけ。

そんなんじゃうまくいかないんだ。例えば『橋本環奈と結婚した〜い』なんてアホがいたとする。こいつがうまくいかないのは、別に不細工だからでも高望みだからでもなく、土俵だけしか見てないわけですよ。そんなんじゃうまくいくわけないんですよ。

当たり前だよなあ。

当たり前なのに、就活になると途端にそれと同じことを言う奴がホイホイ出てくる。じゃあ分かった、任天堂/バンナム、サイゲームズを受けたいのは良いけど、その会社の特性とかニーズって知ってる？って話よ。大抵答えられんのよ。

今回は例として大きな会社を言ったけど、その会社が何に強みを持ってて、自分はどういう面で貢献をしたいのかという所がはっきりしてないと、戦略も立てられんのよ。

とはいえ、会社の細かいところまで知るのは大変。だからまずは手始めに『何系を目指そう』くらいでいいから、イメージしてみよう。

- CGが凄いところでシェーダとか書きたい
- AIが凄い所でディープラーニングって奴で何とかしたい
- とにかくたくさんのゲームを作りたい
- メモリとかスレッドとかシステム周りをやりたい
- ゲームエンジンを作りたい

と言ったところから固めて来ると自分の方針も決まってくるし、どういう所を重要視してる会社かを考えながら探すとうまくいきやすい。こういうのを知りたい人は就職年次の担任の先生とか就職決まってる先輩に聞いてみよう。

で、前述の何系って話だけど、もっと大雑把に言うなら

- とにかく数をこなそう
- 技術力をつきつめよう

てな話になると思います。ゲームを沢山作ればそれだけ技術力も上がると思いますが、浅く広くって感じになるかなと思います。
逆に技術力を突き詰めるってのは、深く深くって感じです。深く深くの場合はちょっと気を付

けておかないと、自分に向いてなかったり会社のニーズが無かったりすると大外れになるので、それも就職年次の先輩とかに相談するといいと思います。

数をこなす人は 10 カ月の間に何本作れるのか…それを考えてください。技術力に自信がないなら、小さなゲームを沢山、とにかくたくさん作るのがいいでしょう。

あと技術系の人にお勧めなのは、CG の見た目に関わる部分を突き詰めるのはお勧めです。何故かというと、ぱっと見が派手だと企業の人の目に留まりやすいからです。何だかんだ言っても売込みってだいじなので。シェーダ書けるようになってポストエフェクトとかできるようになるとかなり強いです。

あと AI 系の人にちょっと言っておくけど、よほどのことがない限りディープラーニングはやめておいた方がいいです。基本の理屈がそもそも高難易度なので、最初は経路探索(ダイクストラ法とか、それ以前の計算幾何学とか)とか、それでも難しければ AI というよりきめられたパターンで動くもの。それも状態遷移を用いて動かすものから作った方がいいでしょう。
よく AIAI 人工知能っていう人は何か勘違いして失敗することが多いです。思ったよりもややこしく、概念そのものを勘違いしていることが多いです。

とはいえ、たぶん大半の人は数をこなした方がいいでしょう。つまるところ、技術を突き詰めるよりもたくさん作った方がいいと思います。その中で何かのめり込めるテーマがあったら、それを突き詰めてもいいでしょう。ただし、ニーズが全然ないものを突き詰めても意味がないので、自信がない人は誰かに相談しましょう。

その際にきれいなプログラミングを心がけて欲しい所ですが、それで手が止まったら元も子もないので、まずは沢山作って、企業に出す前に「リファクタリング」かましましょう。

あと、企業の人の目に停めてもらう回数が多ければ多いほど有利になりますので、魅せるためのデザイン的な所は軽くでいいからお勉強しておきましょう。

あと、できるだけコンテスト等に応募したり、twitter 等で活動を公開したりしてとにかく露出を増やしましょう。それが現代の戦略ってもんです。

まぁ色々と話してきましたが、この残り 10 カ月をどう過ごせばムリなく最大効果を得られるのかを考えながら行動しましょう。

それをやらない人は多分 10 か月後に後悔すると思います(就職年次の先輩の半数が後悔しています)

技術へのアンテナについて

皆さんはゲームプログラマを目指しているので、アンテナを技術に向けて立てておきましょう。

そもそもゲームでどういう技術が使用されているのか？どういう技術が必要なのか？どんな本やサイトを見ればいいのか？
クソ下らねえ芸能人とか Youtuber のゴシップ見てる暇があったら、そういうのにアンテナを張ってください。というかここにいる以上、そういう生き方にもう片足突っ込んでる自覚を持ってください。

ちなみにアンテナの参考になるのが
Qiita
https://qiita.com/

SlideShare
https://www.slideshare.net/

SpeakerDeck
https://speakerdeck.com/

ゲームつくろー
http://marupeke296.com/GameMain.html

Cedil
https://cedil.cesa.or.jp/

あと twitter つよつよエンジニア系のひととかフォローしとこう。でもあんまりツヨツヨばかり見てると自分の未熟さに嫌気がさすので、ほどほどに。

授業の流れについて

で、ここまで色々と話してきましたが、この授業自体は、皆の開発の面倒を見るという感じで

はなく、様々な開発のテクニックを伝授するものです。なので、それをどう活用するかは皆さん次第だし、前述の戦略を自分で立てて、役立つと思ったらメッチャ利用してほしいし、そうでないと判断したら、単位を取るためだけにこなしていけばいいかなと思います。
プログラミング系では多分この学校では一番利用者が多いであろう DxLib で説明していこうと思います。ですが、他の環境でも応用できる話にする予定です。あくまでも DxLib を使うというだけ。

あと、昨年の授業で「他校が DxLib で3D やってるのにビビっちゃった」学生さんがいたので、DxLib で 3D を軽くやっとこうかと思います。本当に軽くね。

で、流れですが(ある意味ここがコマシラバスね)
1. まず開発環境 VisualStudio について(設定とかデバッガとかコマンドラインとか)
2. Git とか GitHub とかについて(チーム制作以外でも)
3. DxLib 使いつくしてる？DrawGraph くらいしか使ってなくね？
4. DxLib で簡単なポストエフェクト(シェーダとか使わない)
5. DxLib で簡単じゃないポストエフェクト(シェーダ使ってみる)
6. DxLib で 3D やってみる
7. ちょっと特殊な当たり判定(題材は DxLib を使うが…)

こんな流れにしようかと思います。週 2 コマのはずなんで上の 1 単元に 4 コマ使う感じで…みんなのレベルが高ければここに書いてる以上の事をやると思いますし、皆がボンクラーズなら全然進まないかなと思います。

あと余裕があれば合間合間で「色の話(効果/組み合わせ)」とか「フォントの話」をしようかなと思います。

どこまで進むかは皆さん次第です。せっかくだからなるべく沢山僕から吸収してしまいましょう。

環境とか VisualStudio とかコマンドについて

環境変数

意外とこれ知らない人が多いんで確認しておきますが、環境変数って知ってます？
3 年生はすでに分かっていると思うのですが、2年生にとってはこれからお話しすることは初耳かもしれません。

「DxLib のフォルダは C:¥DxLib や D:¥DxLib とは限りません」

というか自宅で作業したことのある人ならわかると思いますが、DxLib のライブラリも自分でダウロードすんだよ当然だろ。というのが2年生以降の話です。一応 PC をセットアップした時点で僕が学校内の全 PC の C か D の直下に置きましたが、それができない環境もあったりしますので、

「DxLib をどのフォルダに置いていても環境設定さえすればどの PC でも同じように使える」ということをお教えします。めんどうですよね。

なぜこういうのが必要かというと、就職活動などをする際に、相手方は C や D の直下に DxLib のフォルダなんてまず置きません(もっと言うとダウンロードもしないと思います)。

あと、新2年生にとってはライブラリのリンク経験は DxLib くらいしかないのかもしれませんが、今後のプログラミングをやって行くと分かると思いますが他にもいくらでもあります。

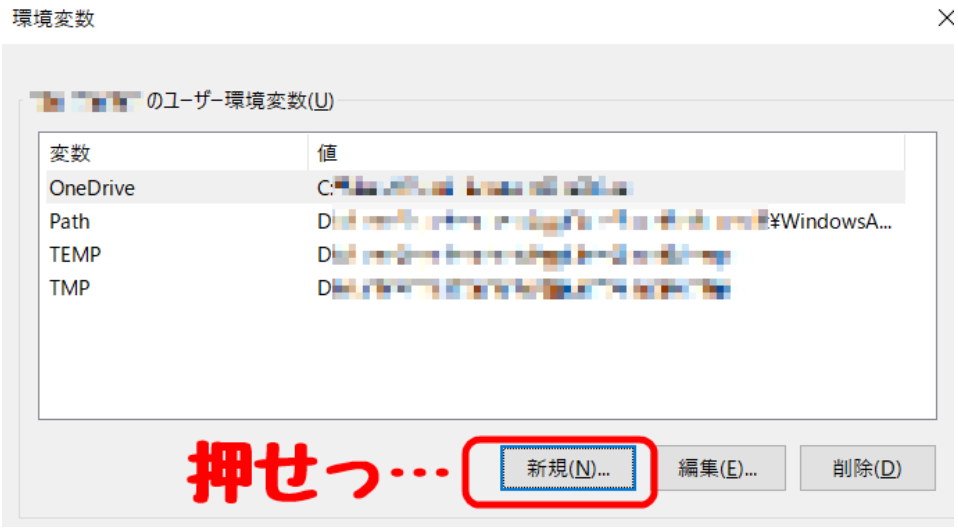
というわけで、今後はライブラリのフォルダを直で書くのはやめましょう。1年生の頃はね、覚えることが多かったからこういう細かいことは教わらなかったと思います。頭がフットーしちゃいますからね。

① まず自分の PC の DxLib のフォルダの場所を確認します。

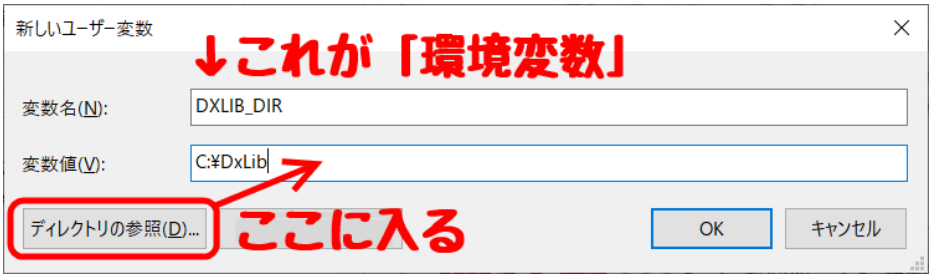
② 次に Windows 左下の検索バーに「環境変数」と日本語で書き込みます

③ そうすると2つくらい候補が出てきますが「システムではない ほう」を選択します。

④ そうすると「環境変数」のウィンドウが立ち上がりますので、上下に分かれている画面の上側で「新規」というボタンを押します。押せよ!絶対押せよ!



⑤ そうすると新規で『環境変数』を作るためのウィンドウが出てきます。今回の環境変数はディレクトリのためのものなので『ディレクトリ』ボタンを押して、DxLib が置いてあるディレクトリを選択してください。
また、上の段はそのディレクトリを表すマクロ変数みたいな感じになるので、そこに DXLIB_DIR と書いておく。終わったら **OK を押す**(☆右上の×を押すなよ!絶対押すなよ!)



⑥ OK 押すと元のウィンドウに戻る。この時点で環境変数が出来上がっていると思うだろう？素人はまんまと引っかかる。

ところが大間違いなんだ。よくやるやつが右上の×を押しちゃうやつ。違う…っ!!

ウィンドウを閉じるときに反射的に×を押したくなる気持ちはわかる…っ!!しかし…それが罠っ…!!!巧妙に仕掛けられた悪魔的…罠っ!!今までの作業が台無しになってしまう。必ず OK を押すようにしよう。

⑦ さて、これで話は終わりではない。コマンドプロンプトを立ち上げてくれ。何ィ？コマンドプロンプトを知らないだぁ!?お前ここをカルチャーセンターと間違えてんじゃねえのかァ？

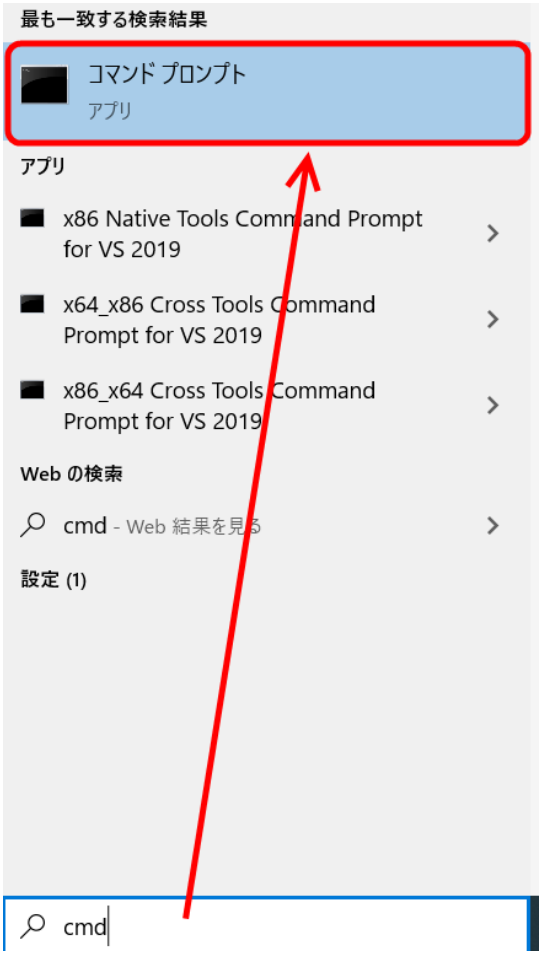
コマンドプロンプトというやつは、システムコマンドを打ち込むためのものです。Linux とか使ったことのある人なら Linux コマンドは知っていると思うけど、ls だの cp だの cd だのというコマンドで OS に命令を出したり情報を引き出したりするためのものだ。

出し方はいたって簡単。ご注目!!いきますよ～よく見といてくださ～い。

先ほど環境設定の時にやったように左下の検索バーに"cmd"と3文字打ち込んでください。

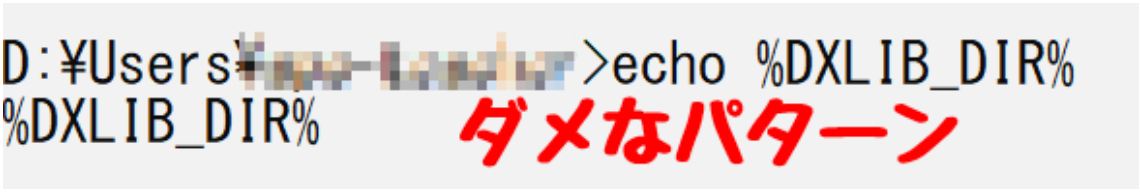
いいですかー、しー、えむでぃー、ですよ～。

はいこれで、立ち上がります。見覚えのある人もいるかと思います。デフォルトが黒背景の白文字なので、説明では見づらさ軽減のために設定で白背景の黒文字で表記しますね。



⑧ コマンドプロンプト上で"echo %DXLIB_DIR%と打ってエンター。まともに設定できていれ

ば、完全なパスが表示される。表示されなければ設定に失敗してんだよもっかいやんだよあくしろよ。



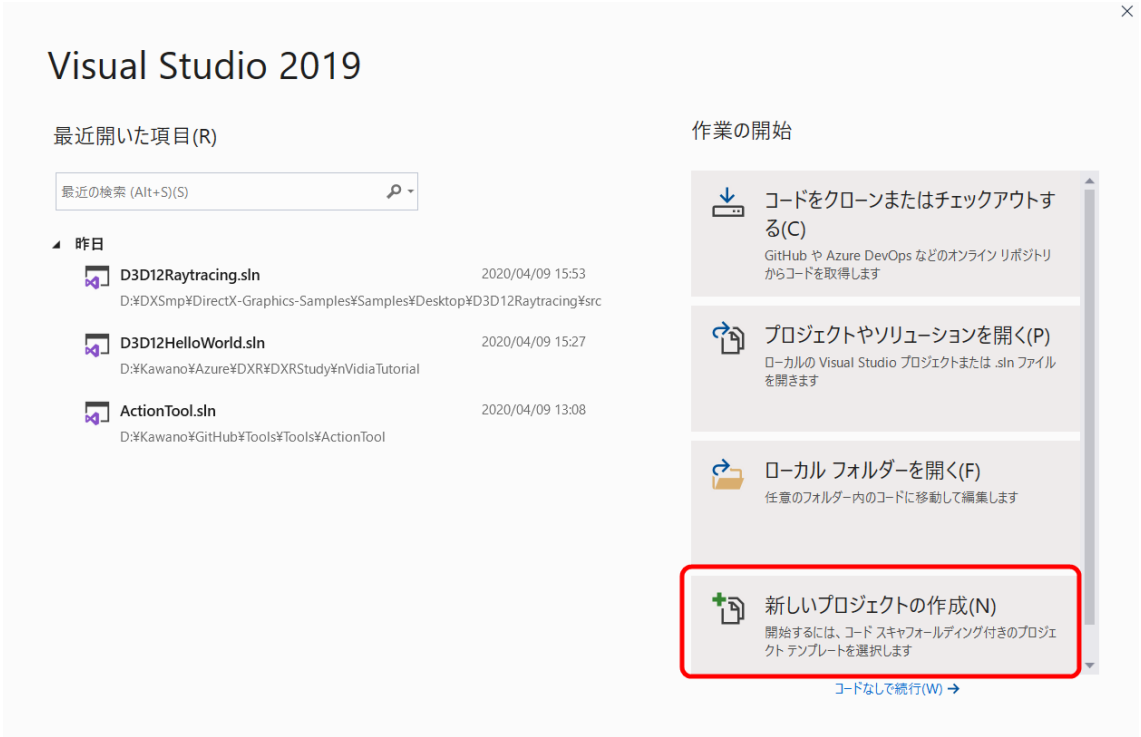
うまくいっていればフルパスが表示されます。なお、やり直す際はいったん全てのコマンドプロンプトを落としてもう一度立ち上げなおしてください。OK なら

このようにフルパスになります。これで設定できていることが確認できました。

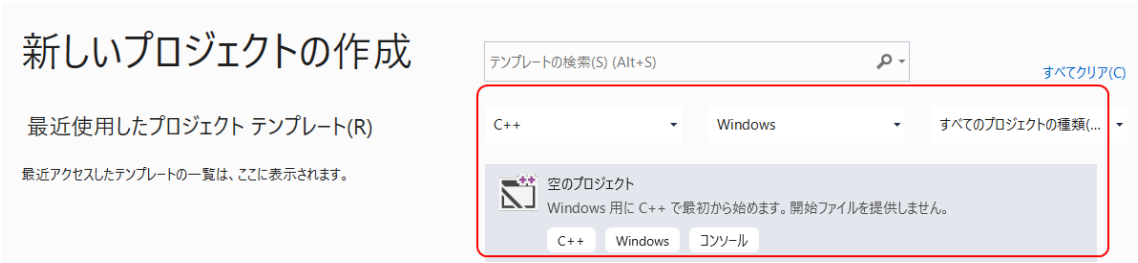
⑨ 次にもし VisualStudio を立ち上げているのであればすべて終了してください。一つでも立ち上がっている状態だと、この環境変数の変更の影響を受けないからです。

⑩ そして VisualStudio を再び立ち上げて

新しいプロジェクトを作成します。



ここで作るのは「空のプロジェクト」(C++ Windows Console)です。

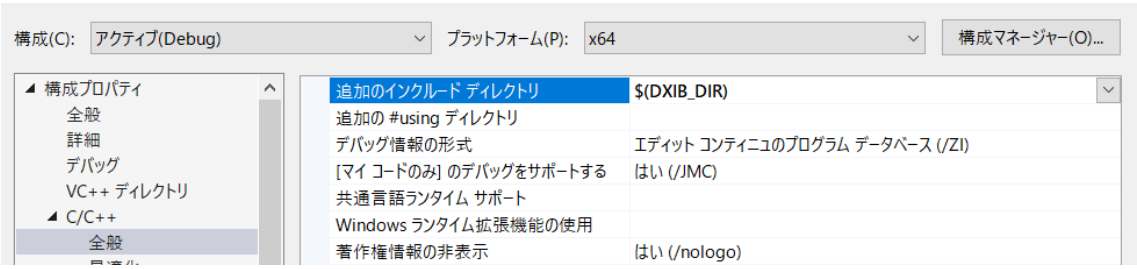


⑪ ソースコードとして main.cpp を追加してください。で、以下のコードを追加します。

```
#include<DxLib.h>

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int) {
	DxLib::SetGraphMode(1280, 720, 32);
	DxLib::ChangeWindowMode(true);
	DxLib::SetWindowText(L"Ninja Sprit");
	if (DxLib_Init()) {
		return -1;
	}
	DxLib::SetDrawScreen(DX_SCREEN_BACK);
	while (DxLib::ProcessMessage() == 0) {
		DxLib::ScreenFlip();
	}
	return 0;
}
```

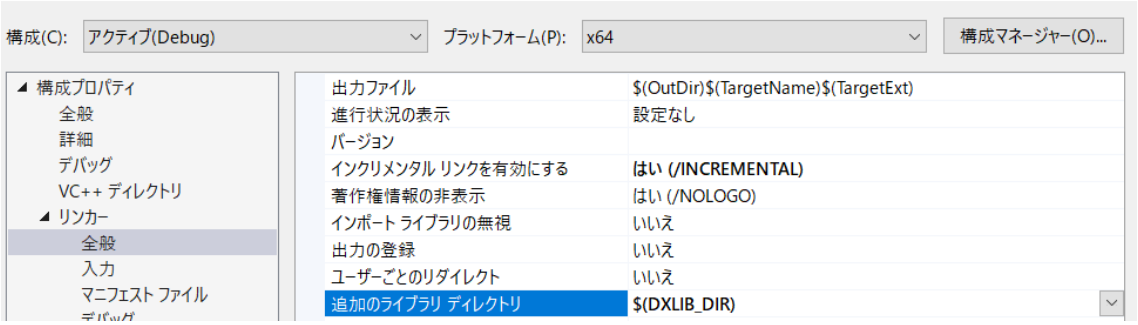
まぁ、言うてもこんなん本質にはかかわりないので、写経しようがなんしようが好きにすればいいです。

⑫ このままでは DxLib へのインクルードパスもリンクパスも通ってないので、動かすことができません。そこでプロジェクトの設定に移ります。設定のところで



追加のインクルードディレクトリで$(さっき作った環境変数名)と書きます。この文字列が先ほど作った環境変数が表すディレクトリパス文字列に置換されます。

⑬　次にリンク部分にも同じことをやります。



追加のライブラリディレクトリに$(環境変数名)とします。

はい、一応これでスケルトンプロジェクトが立ち上がるはずです。簡単なのでさっさとやりましょう。全然プログラミングと関係ない部分です。
なお、インクルードとリンカの部分を話しましたが、そのうち「コンパイル」と「リンク」の意味についてもお話しします(大事なことです)。

で、実行しようとするとリンカエラーが起こることがあります。その場合はサブシステムが「CONSOLE」になっているので「WINDOWS」に変更してください。(なお、サブシステムはリンカーのシステムの項目を見ればあります)

あと、今回の授業では 64 ビットで統一しようと思います。皆さんはどちらを選んでもかまいませんが、もし 32 ビットを選択して起きたトラブルに関してはご自分でご対応ください。よろしくお願いいたします。

ともかく実行するとこんなウィンドウが出ます。

出ましたね？次行きます。

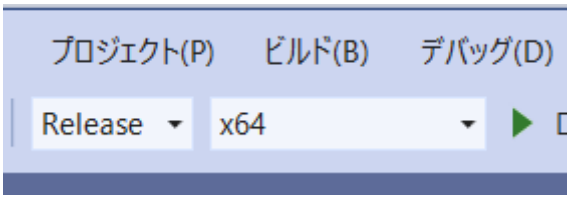
ちなみにコマンドラインのコマンドの echo は環境変数の真の姿を表示するものです。これは後述します。

じゃあまず、そもそもの道具の VisualStudio についての理解を教えてくれるかな？

VisualStudio のプロジェクト設定

プロジェクト設定の前に…

そういえばプロジェクト設定の話の前にさ…



これ、きちんと気にしてます？これって、デバッグでもリリースでもともかく実行ファイルを作るときに指定するものなんだけどね？そもそもデバッグとリリースの区別を知らん人がいるっぽいので言っておくと左のプルダウンメニューには通常「Debug/Release」があって、

- Debug はデバッグするための exe を作るモード
- Release は実際に配布するアプリとして exe を作るモード

なのよ。なので**コンテストに出す時とか、企業に提出する時には Release モードでビルドしたものを提出して**ね。この辺キチンとしてないと「そんなのも知らないのプギャー!」ってなって中身も見られずに落とされたりするからね。

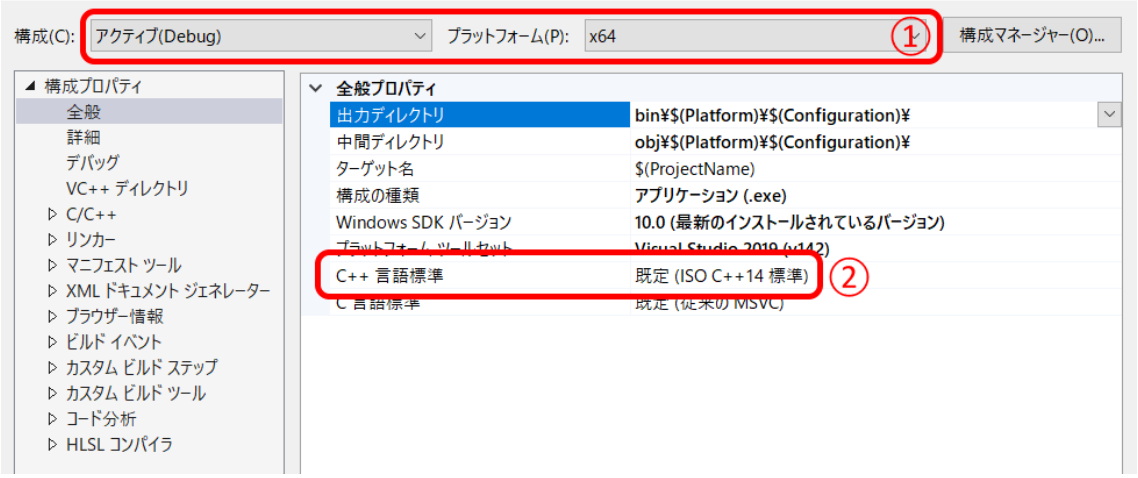
あと、その右側の x64 ってあるけど、普通は x64 以外に Win32 か x86 ってあるけど、これは古いＰＣ向けって事ね(32bit マシンの事)今時 32bit しか受け付けないってＰＣの会社もないだろうし…というかそんな今時 32bit マシンのＩＴ人権のない会社にはいかない方がいいです。

なので、基本的には「x64」にしておきましょう。ここをまずは「Debug」「x64」にしておいてください。後述するプロジェクト設定はここに合わせてやることになります。

なお、面倒ですが、きっちり設定を使用と思ったら Debug の設定、Release の設定の2つを設定しなければならないためそれは把握しておきましょう。

プロジェクト設定

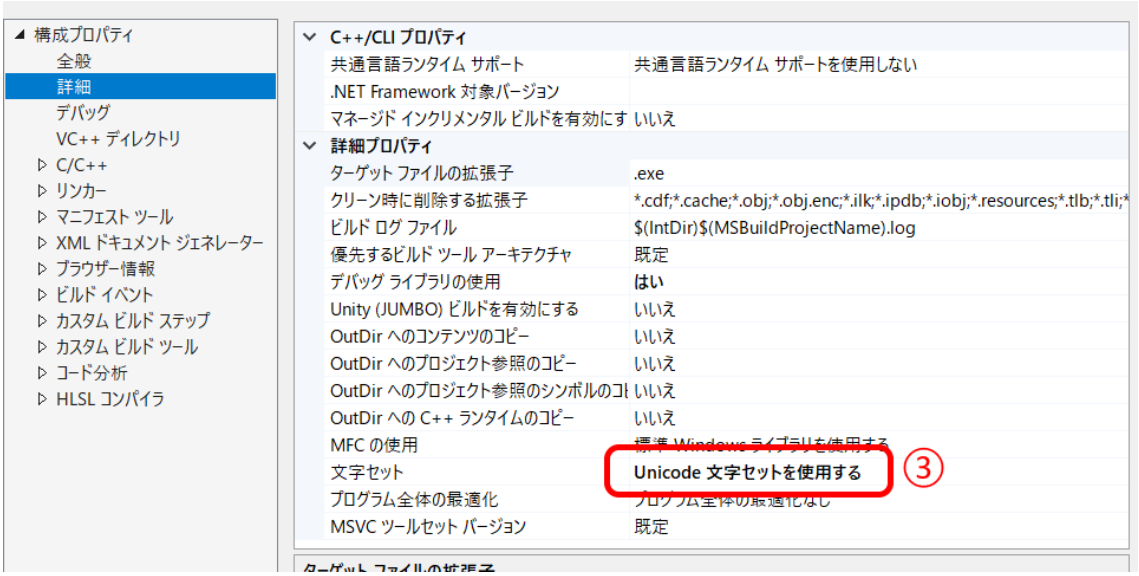
さてさて、プロジェクト設定の中身ですが開くとこんな感じになるかと思います。



まず①は必ず確認しましょう。今設定中のものがDebugなのかReleaseなのかx86なのかx64なのか…ここを間違えるとせっかく設定したものが実行時に反映されておらず、あれ？あれ？ってなります。

次に②ですが、これはＣ＋＋の機能をどれ標準にするのかを設定します。今のVisualStudio2019ですと、Ｃ＋＋14が既定になっていますが、現代はC++17が標準なので、ここはC++17にしておくことをお勧めします。

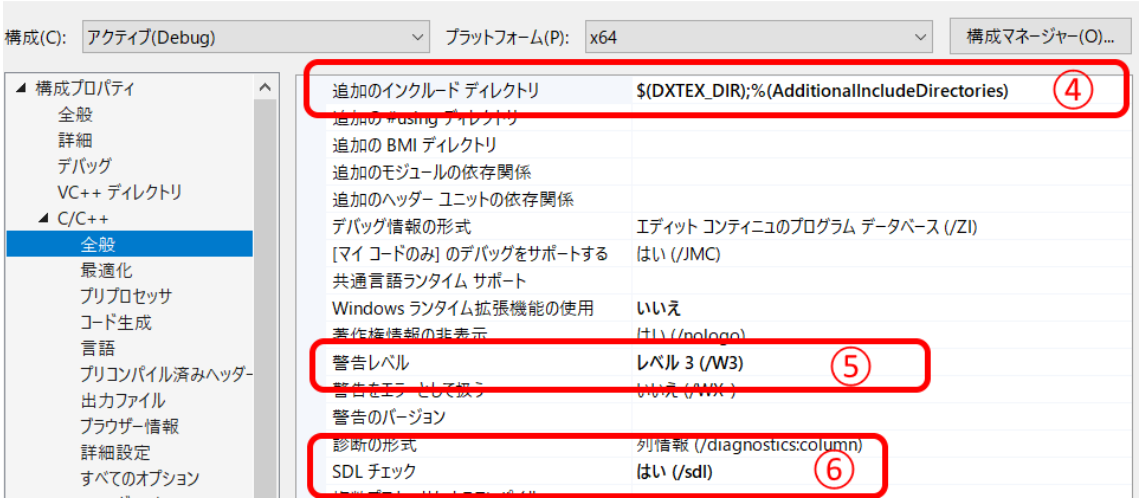
次は「詳細」をクリックしてください。ここは見るのは③の部分だけでいいです。



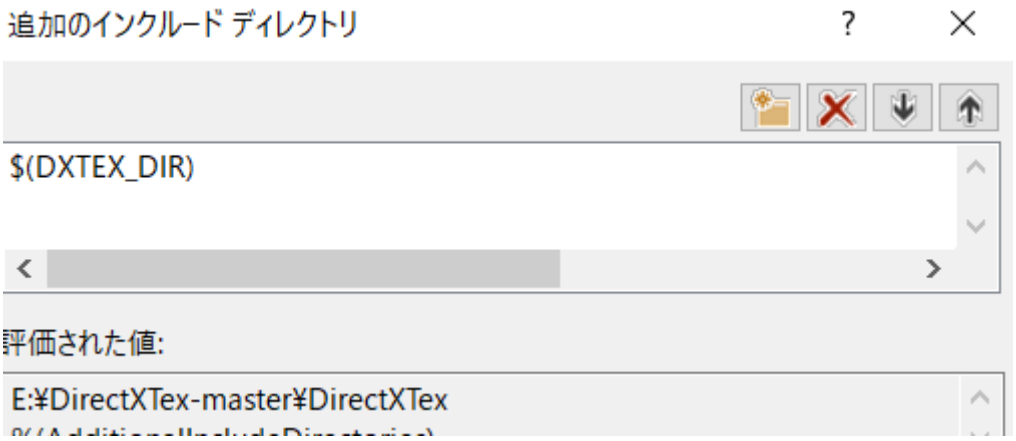
Unicode文字セットとマルチバイト文字セットのどちらかを選択できます。

プログラミング初心者は「マルチバイト文字セット」がいいと思いますし、今後の拡張性を考えるなら「Unicode 文字セット」の方がいいと思います。

文字セットの話はちょっとややこしいので、ここでは 2 種類あるという事だけ把握しておいてください。



④はみんな大好き追加のインクルードディレクトリですね。おなじみですね。上では環境変数を入れてますが、④の枠の右にあるプルダウンメニューから編集と押すと



が出てきます。環境変数の DXTEX_DIR の中身が見えていると思いますが、ここでパスがまちがってないかどうか確認しましょう。

次に⑤ですが、これは警告レベルです。数値を上げれば上げる程警告が厳しくなります。セキュアなプログラミングをしたい場合には上げまくりましょう。

⑥もセキュリティ関するものです。ここを「はい」にしているとメモリアクセス系の標準関数

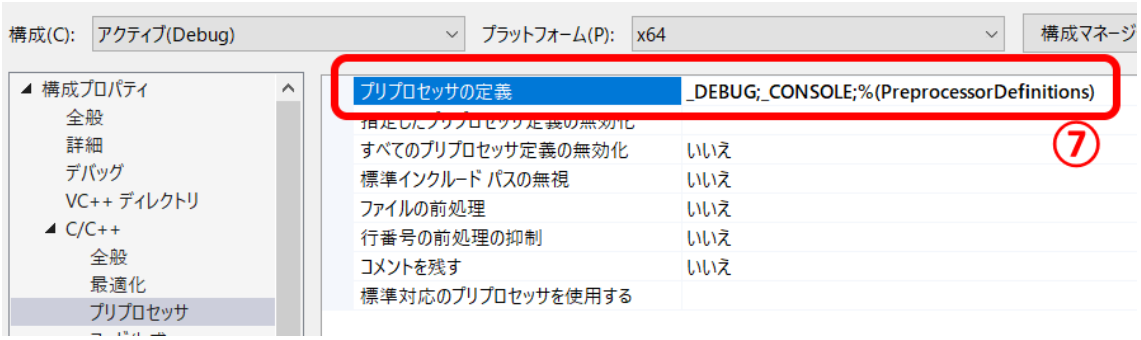
には_s をつけたものを使用しないと怒られます。

scanf←SDL がハイの時は不可。scanf_s とすべき。なんですが、これ MS でしか通用しないし、色々と問題あるので、ぼくは「いいえ」にしておくことをお勧めします。

で、次の最適化の所は、Debug/Release の区別がついてりゃいいので、飛ばしてよくて、割と大事なのはその次の「プリプロセッサ」です。

プリプロセッサの細かい話は後述しますが、大雑把に言うと pre(前の)process(処理)からきてる言葉なので「前処理するやつ」みたいな意味です。

ここでは C/C++言語の中でよく見かける＃〇〇に関わっていると思ってください。
#include
#define
#pragma
#ifdef～#end
などですね。実際に設定画面を見てみましょう。



はい、基本的には↑の⑦…ここしかいじる所はございません。
「プリプロセッサの定義」なんて書いとりますけれどもこの「定義」ってのが define の直訳だという事に気づけば…あとはわかるな？

つまるところここで#define と同じことができるわけ。なんだけどそこまで意識して使ってる人が何人いるのかなって感じもする。

ちなみに上のレイだと_DEBUG やら_CONSOLE やら書いてますが、最近はこれすら書かなくなったかな。デフォルトだと_MBCS って書いてるけど、これは「マルチバイト文字セットを使用します」って事ですね。

通常の DxLib を使用している分にはあまり見る事はありませんが、一つだけ覚えておいた方

がいル「定義」があります。それは

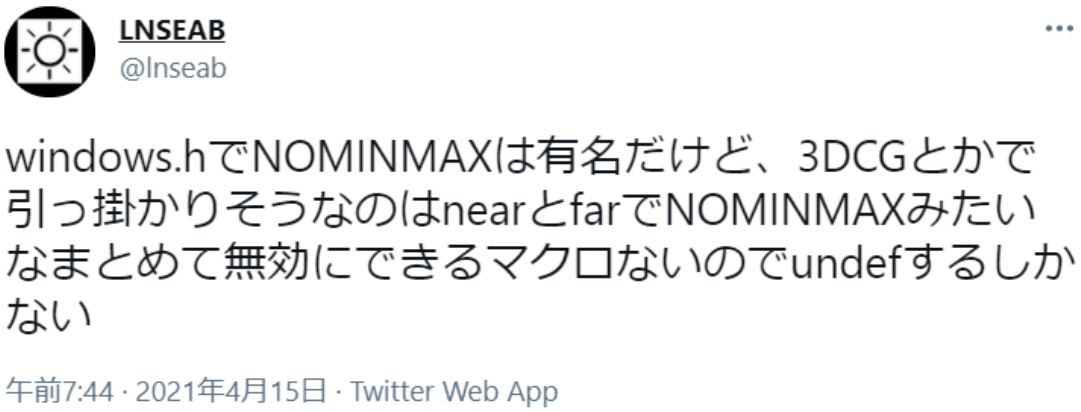
NOMINMAX
（農民マックス！）

です。これは何かというと

Windows 系の開発をしているときに Windows.h のクソマクロ(ファッキュー!!)の影響で min とか max のつく関数が誤動作を始めるのだ。恐ろしい恐ろしい…。

で、そもそもそのクソマクロを無効化してくれる「定義」なのだ。恐らくは Windows.h を作った側もこれが「クソマクロ」だという事は自覚しているのだろう…。

ちなみに朝にこれを twitter で呟いたらこういう事をつぶやいてる人がいまして…



あー…near far もそうなのか。本当に define マクロは害悪だよ…と思いました。
define マクロを知らない人のために軽くだけ説明しておくと、define マクロとは define の機能を関数的な感じで使ってしまおうというわけだ。

例えばこのように定義する

```
#define min(x,y) x<y?x:y
```

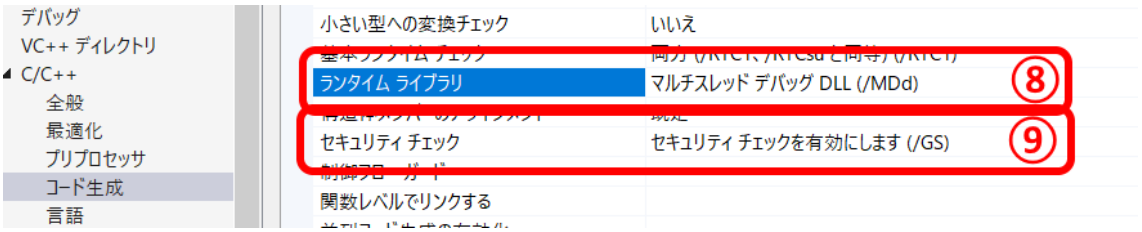
これが何やってるか分からない人はもう一回 C 言語を勉強しなおしましょう(三項演算子だよ)ね。
まぁ、それはともかくこいつは副作用がヤバくてプログラム文中に min(〇,〇)という文字列を

見つけるや否や上の展開をしてしまうのだ。これの何がヤバいって

oremin(a,b)だろうが noumin(米,人参)だろうが問答無用で展開しやがる困った奴なのだ。ちなみにここで「いやでもそれをやっちゃうと min 関数と max 関数使えなくなるんでしょ？」と思われるかもしれないが、心配ご無用!!

C++には安全な min と max がございますのでそれをお使いください。std::min(a,b)と std::max(c,d)というふうに。

次にコード生成です



これはコンパイル時にどのようなコード(ネイティブ機械語)を生成するのかを聞いてきます。意味が分からないと思いますので、気を付けておくべき設定を 2 か所言います。

⑧は、Windows の OS に関連した VisualStudio 標準ライブラリをどうリンクすべきかで機械語が変わってしまうのだ。意味が分からんと思うので、言っておくと⑧は DLL って書いてる奴と、DLL って書いてない奴がある。
で、大抵の場合は **DLL って書いてないほうを選択した方がいい。**何でかというと、DLL はダイナミックリンクライブラリ(動的にリンクするライブラリ)を使用するという意味で、今使用している VisualStudio に関連した DLL も添付して配布しなければならなくなる。

つまり exe 単品では動かない設定なのだ。何でこういうヤヤコシイ事をするのかというと、標準の物を DLL 化しておけば exe のサイズが小さくて済むからだ。しかし最近の潤沢な環境とそれぞれの環境の複雑さを考えると DLL 化はデメリットの方が大きいと言える。

しかし気を付けておかねばならないのが、この DLL がデフォルトという事だ。じゃあ片っ端から DLL じゃないやつにして行きゃいいかというと、そうでもないのだ。何かというと、使用している外部ライブラリが DLL つきのやつでコンパイルされている場合、リンカエラーを起こすのだ。

このため DxLib ではどちらでもリンクできるように細工を施しています。が、世の中そんな気を使うライブラリばかりではないので、自分が使用するライブラリが DLL 系じゃないかどう

かだけは確認しておきましょう(最近は大抵のフリーの便利ライブラリの場合 GitHub でオープンソース化しているため、自前でコンパイルすることになりますが、そのライブラリのコンパイル時に、この DLL 使ってるかどうかの設定は合わせておかなければなりません)

次に⑨の「セキュリティチェック」ですが、こいつは基本的に ON にしておきましょう。
これはランタイム時にバッファオーバーランを検出するとクラッシュしてくれます。え？クラッシュ？やめてよと思った人はまだ甘い。

バグがあるならさっさとクラッシュした方がいい!!事故現場に近い方が直しやすいのです。

VisualStudio のデバッグ機能

うーん。意外と VisualStudio のデバッグ機能についてきちんと知らない人が多いみたいなので、んで、これ知ってるとバグ起きた時の対処の時間が大幅に短縮されるので、簡単なはずだけど、もしかしたらちょいと難しいかもしれないけど、パパパッと説明しますね。

ブレークポイント

デバッグの基本のブレークポイント…これ、正しく扱えてますか？当然ながらブレークポイントをソースコードの特定の行に置けば、そこでコードの実行は中断されまぁす!

ブレークしてからですが、そこで F5 を押せば中断したところから再実行されます。

ステップ実行

F10 を押せばステップ実行と言って、1 行ずつ進めることができます。ただし関数の中には入らず、関数の呼び出し行で F10 を押せばその関数が実行されたことになって、次の行に進みます。

ここで F11 を押せば関数の中に入ります。ただし常に F11 を押してるといちいち関数の中に入るので面倒ではあります。例えば

```
Function(Func1(),Func2());
```

こんな関数が書いてある行で F11 を押せば関数の出入りを 3 回繰り返すのでウザいです。こういう場合は、この関数が書いてある部分にブレークポイントを置いておいて、一旦止めて

おいて、怪しいと思う関数の中にまたブレークポイントを置いてF10実行した方がいいです。

ちなみにF10をステップオーバー実行、F11をステップイン実行と言ったりします。いちおうどちらも「ステップ実行」と言います。

ここまでは知ってる人がほとんどなんじゃないかなと思います。知らなかった人は覚えてね。
で、次は半数くらいの人がびっくり!!するだろう

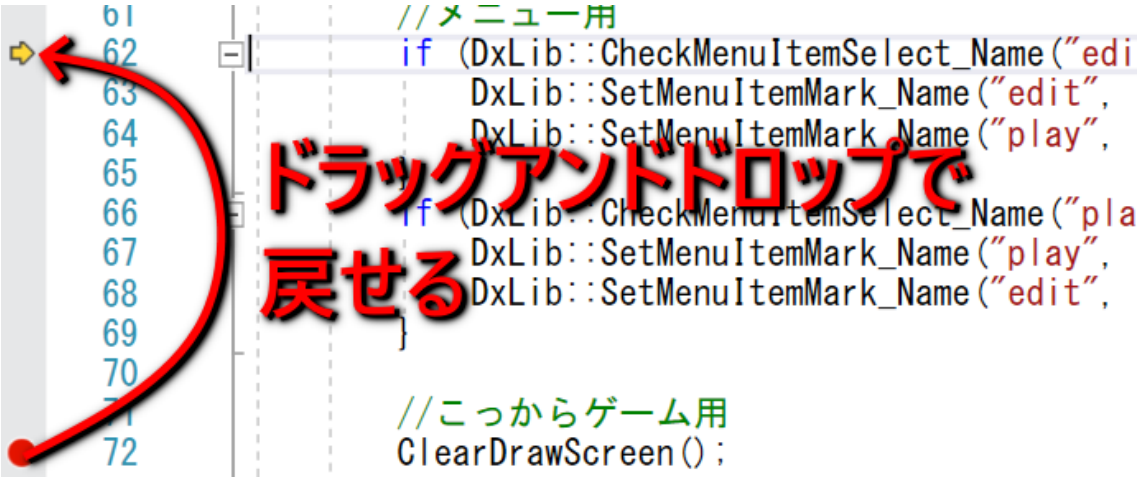
デバッグカーソル(?)をドラッグできちゃう

知ってた？
あっ、そう!!!まぁ、それはそれとして解説しよう。

まず、どっかでブレークポイントしかけて、止まってる状態にしてみよう。そしたらデバッグカーソルが黄色で表示されているはずだ。

ここに対して、自分のマウスポインタを合わせてみよう。

驚くべきことに…こいつをドラッグアンドドロップできるのです。嘘だと思うんならドラッグアンドドロップしてください。



ドラッグアンドドロップで戻せるし、逆に進めることもできます。あと、目的の行でコントロールキーを押してると黄色い矢印ボタンが出るので、それを押してもらうとデバッグカーソルがそこに飛びます。

どういう時に役に立つのかと言うと、F10 でステップ実行して、うっかり目的の行を素通りしちゃうことがありますが、これで戻せばいいわけですね。

ただし注意点として F10 で一度実行されてる処理は、カーソルを戻しても「なかったコトに」はできません。そらそうだ。

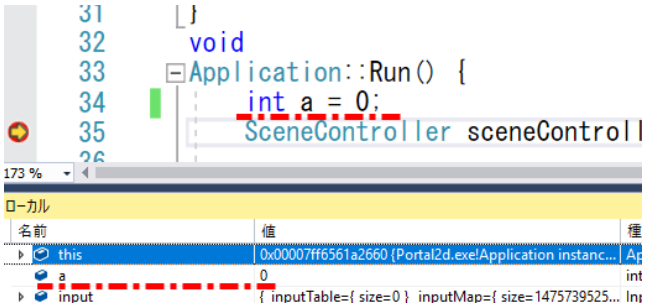
この機能を用いると、**デバッグ情報の信頼性が著しく低下する**ので、使うのはなるべくやめておいた方がいいとは思います。でも便利なので、デバッグの信頼性が下がっても大して問題ない時は利用してもいいと思います。

結局使う人のスキル次第。よくこういうのは「教えない方がいい」と言う人がいますが、知ったうえで使わないのと、知らないから使わないのでは意味が違います。知ったうえで、その危険性は認識しておくべきだと思います。

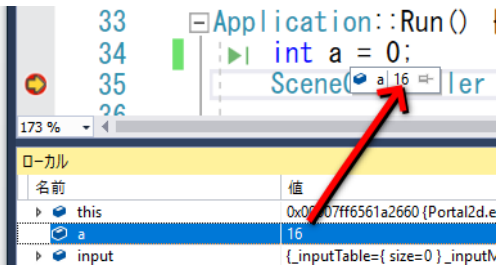
それではそういうデバッグの信頼性を下げるけど、便利な機能をもう一つ…

デバッグ中に値の変更もできちゃう…できちゃう

例えば以下のような場合



当然ながら変数 a の中身は 0 です。この中身を変更できるでしょうか？できますんよ。ローカルもしくはウォッチで a を探すと、名前と値の表がありますが、この「値」に適当な値…16 でも入れてあげましょう。



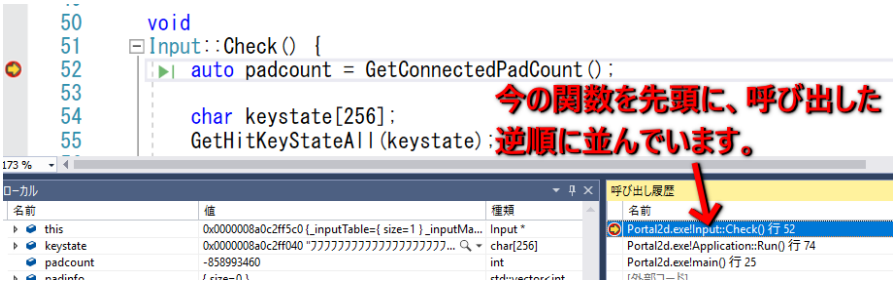
こんなことできます

ま、これもお分かりのようにデバッグの信頼性を損なうので、ホントにテスト的な事をやる目

的以外には使用しないようにしましょう。

呼び出し履歴(コールスタック)

まま、これは分かり切ってると思うんで、軽く流しますが、例えばどこかでブレークポイントさせるか、どこかでアサーション起こすと、当然処理が止まるのですがその時に呼び出し履歴(コールスタック)ウィンドウを表示させると



で、右下のコールスタックの各行をダブルクリックすると、その関数に飛び、さらにはその関数呼び出し時の周囲のローカル変数なども参照できます。非常に重宝します。

発展的ブレークポイント

さて、再びのブレークポイントですが、使い方をもう一歩進めると、非常にバグの検出の役に立ちます。

条件付きブレークポイント

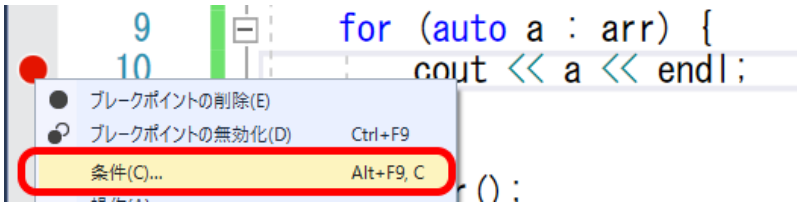
例えばこんなコードを考えてみる。例えばだよ？このコードに対した意味なんてないよ。

```
#include<iostream>

using namespace std;

int main() {

	int arr[] = { 1,2,3,4,5,6,7,8,9 };

	for (auto a : arr) {
		cout << a << endl;
	}

	getchar();
```

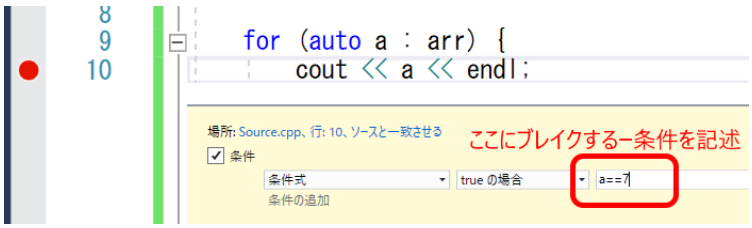
```
        return 0;
}
```

さて、なんかしらの理由で、要素が 7 の時の状況を知りたいとする。君だったらどうするだろう？7 回ループを回す？それじゃあもしこれが 1298 ループ目の状況を知りたいときだったらどうするんだろう…。

まぁ、現実的じゃないね。そんなときに役に立つのが条件付きブレークポイントだ。まずいつものようにブレークポイントを仕掛けてみる。そして、ブレークポイント上で右クリックしてみる。



そして「条件」という項目をクリックすると…



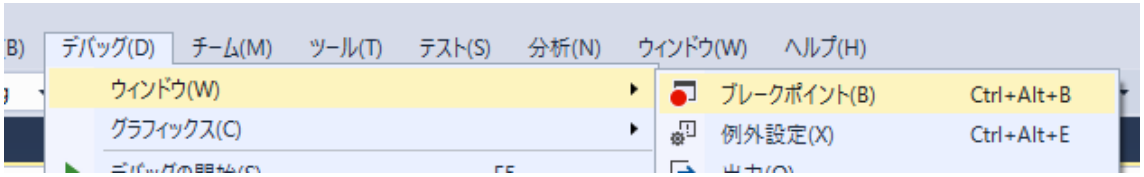
こんなのが出てくるので、ブレイクさせたい条件を記述する。そうすると条件が一致した時だけブレークするため、特定の条件で止めたいときは重宝します。ただし副作用として、条件ブレークを置いている個所は若干処理スピードが落ちます。まぁバグった時にしか使わないから問題ないと思います。

データブレークポイント

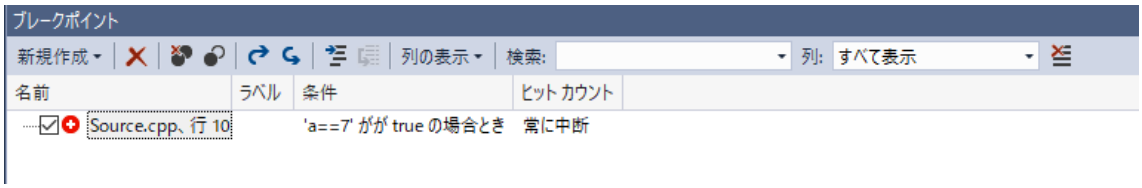
それでは次に、ちょっと難しいのを紹介します。「データブレークポイント」です。
例えば、なんかの値が変化した瞬間を捕まえたいときってありますよね？そういう時に役に立つ機能です。

よくあるトラブルとして、値を代入した覚えもないのに値が変わってるとか、あとは特にグローバル変数とかを使用してる場合ですが、ありとあらゆるところから変更されるため、誰が犯人か分からない…と言うのがあると思います。そういう時に役に立つのがデータブレークポイントです。

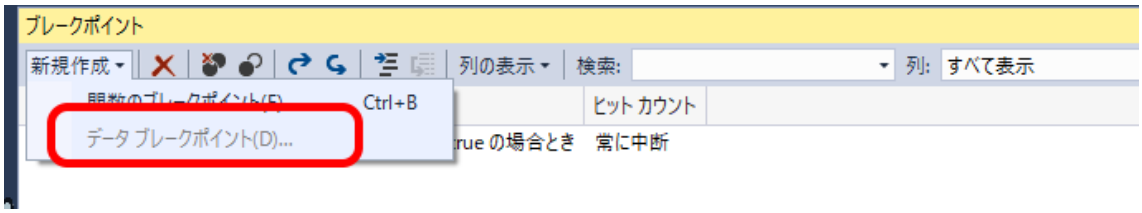
まず、デバッグ→ウィンドウ→ブレークポイントをクリック



はい、そうすると下にこんなウィンドウが出てると思います。



で、この新規作成をクリックするとデータブレークポイントって項目が出てきます。なお、これはデバッグ実行時しか有効ではないので、デバッグしてないときは

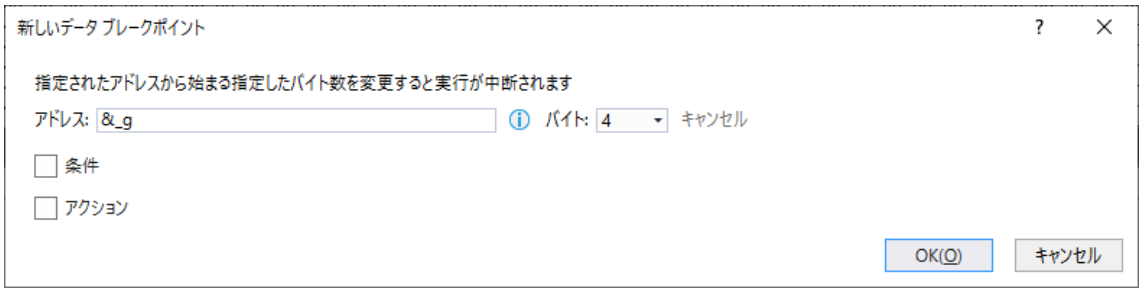


こんな感じで使用できません。なんでかというと、データの置き場所(つまり変数のアドレスなど)というのは、実行時にしか確定しないからです。

で、どうするのかというと、たとえばグローバルに
int _g=10;
なんてのがあったとします。

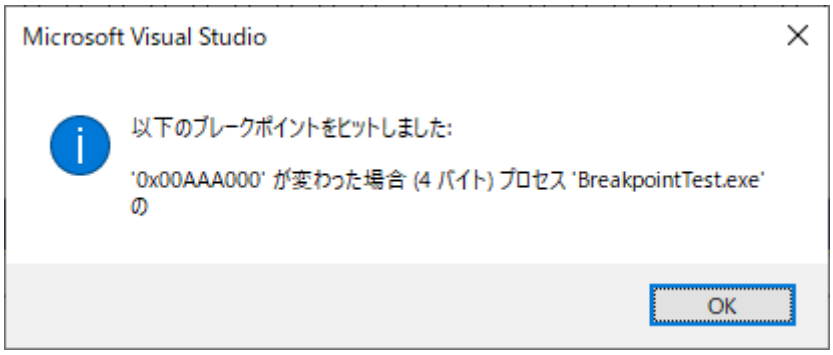
で、誰かがどこかでこれを変更しているのだが、誰だかどこだかわからない。それを知りたいとき…まぁ、実行時にしか分からないので、main 関数の最初にでも普通のブレークポイントを置いて止めます。

そうしたら先ほどのデータブレークポイントが使えるようになってるので、選択します。そうすると



こういう画面が出てきますので、アドレスの部分に対象となる変数のアドレスを入れます。くれぐれも間違えないようにしてほしいのですが、アドレスです。なので上の例では_g では

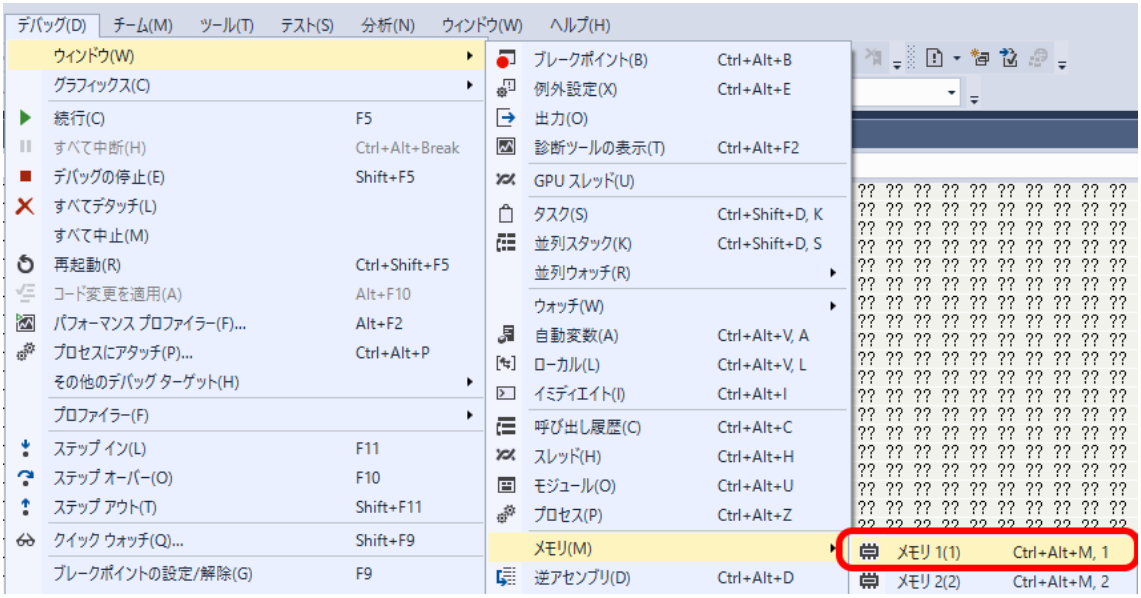
なく、&_g としています。そうすると、変更されたタイミングで



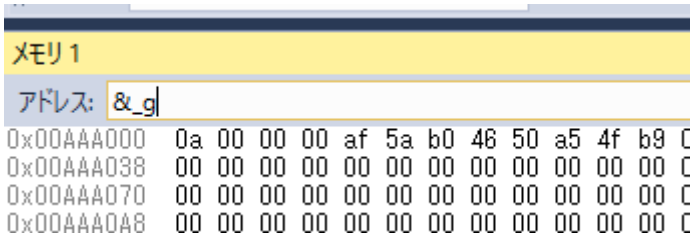
みたいなメッセージボックスとともに処理が中断されます。
まぁ、これの便利さは、C++でクソみたいなバグに悩まされないと、このありがたみは分からないともいますが、このクラスで開発するなら、まぁそのうちお目にかかれます。

メモリの中身をみる

これまたデバッグ時にしか見れないものですが、



こんな感じでメモリを選択すると現在のメモリの状況が見れます。例えば先ほどの&_g のアドレスを入れると…



メモリの中…アドレスさえ渡してあげれば、現在のメモリの状況を 16 進数で見ることがで

きます。これが何の役に立つのかというと、バイナリファイルの読み込みや、なんかしらのバイナリ計算の結果やアドレスの状況を知るために使えます。まだまだ高度すぎるかもしれませんが、そのうち役に立つと思います。

出力ウィンドウ

最後に忘れちゃいけないのが「出力ウィンドウ」です。これ意外と知らん人が多かったから描いておきます。
VisualStudio には「出力ウィンドウ」というものがあり、最初からウィンドウがある事も多いのですが、出てないときは

デバッグ→ウィンドウ→出力

出力ウィンドウを開く事ができます。コンソール対象時はどうせコマンドラインに文字出力できるので、そっちを使えばいいのですが、ウィンドウアプリを作るときはコマンドラインがないので、こちらに出力しましょう。

文字列の出力には
OutputDebugString
という関数がありますのでそれを利用します。ただしこの関数の役割は「文字列出力」のみですので、printf や DrawFormatString のようにフォーマット文字列を出力する機能はありません。どういうことかというと、数値などを出力することはできないということです。

このためなんかしらの数値情報を出力したければ sprintf や stringstream のお世話になる事になります。これも知らん人が多いっぽいので、後々解説します。

あと、DxLib やら他の外部ライブラリは結構情報をここに出力しているので「なんか知らんが壊れた」みたいな時はまず出力ウィンドウになんかログが出てないか確かめておきましょう。

あと最後に注意点ですが、この出力ウィンドウへの出力は結構コストがかかりますのでデバッグ時以外には利用しない事と、毎フレーム固定で何行も出力するのはやめましょう。処理落ちの原因になったりします。

他にもスレッドだのなんだのありますが、それはもうちょっと先(就職してから実際のややこしいバグに悩まされてから)でいいでしょう。今回はこのくらいで十分だと思います。

ビルド…コンパイル/リンク

多分 2 年生になったばかりの皆さんはプログラム書いたら→デバッグ実行〜ってな感じでやってると思います。それはそれでいいです。全然問題ないしプロがやってる事もあまり変わりありません。
それはそうなのですが、もう 1 ランク先に行くには…というかわけが分からないエラーに対応するためには、あのデバッグ実行ボタンを押した時に何が行われているかを知っておいた方がいいと思います。

コマンドラインについて

新 2 年生はコマンドラインを使った経験もあまりないと思います。そもそもそんなのがある事すら知らんかった!って人もいるかもしれませんが、実は結構使います。

色々と理由はあるんですが、主な理由は
コマンドラインの方が手っ取り早いことがある
バッチファイルを作るため
コマンドラインでしか動かないアプリケーションがあるため

主なコマンド

Git について

Git については別紙を用意してるんでちょっとそっちを見てもらうとして…