浮動小数点型変数で遊ぼっ！

花子

- 誰と遊ぶ？
- それ、どんな子？
- 双子もいるの？　区別つくかなぁ。。。
- どこで遊ぶ？
- かけっこしよっ！
- 限界超えて遊ぶぞっ！　朝までオール？
- でもオールは疲れるよ。。。
- 違う公園も行こー！
- さっきのかけっこでズルしたっしょ？
- 遊び足りない？
- そろそろお寺の鐘もなるし。。。おかたづけ

誰と遊ぶ？

C#ちゃん、VBくんとは遊びません。

Visual C++ 2008だけです。

（Professional Edition 90日間お試し版ですが。。。）

浮動小数点型は・・・

float, double, long double

C言語には　_Complex：複素数型

_Imaginary：虚数型

今回は、実数の浮動小数点型について

それ、どんな子？

言語仕様では、相対的な精度だけ決まってます。

float≦double≦long double

IEEE 754：浮動小数点演算に関する規格

	符号部	指数部	仮数部
単精度	1ビット	8ビット	23ビット
倍精度	1ビット	11ビット	52ビット
拡張単精度	1ビット	11ビット以上	31ビット以上
拡張倍精度	1ビット	15ビット以上	63ビット以上

それ、どんな子？

Visual C++ 2008では・・・

float　　：単精度（32bit）

double　　：倍精度（64bit）

long double：doubleに変換される（64bit）

16ビット版Visual C++では、

拡張倍精度（80bit）

それ、どんな子？

指数部：バイアス（127/1023）を足した値を設定
仮数部：暗黙の1で、精度を1ビット上げる

0.1をfloatにしてみると・・・

0.0001100110011001100110011001100・・・

1.10011001100110011001101×2の-4乗

符号部：0

指数部：-4に127を足して 0111 1011

仮数部：1を取って 100 1100 1100 1100 1100 1101

双子もいるの？　区別つくかなぁ。。。
大丈夫！違う服を着ています。

long doubleはdoubleに変換される

Visual C++ 2008では、

精度は同じだけど、型は違う

```
int main(void)
    long double  ll = 0.1L;
    float        ff = ll;    ⇒ ‘long double’ から ‘float’ への変換です。
                                 データが失われる可能性があります。
    double       dd = ll;   ⇒ OK
}
```

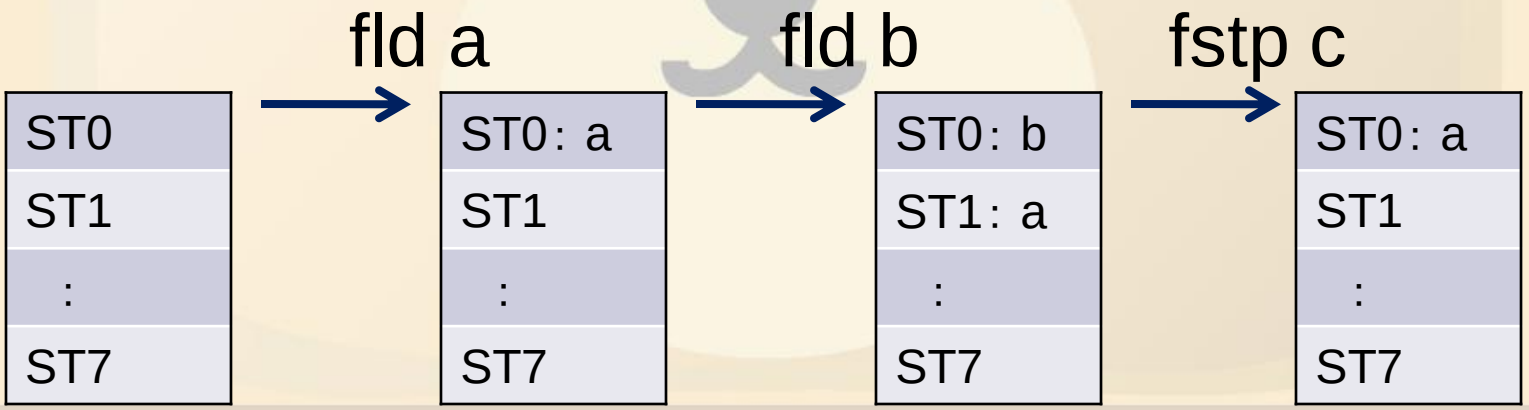
双子もいるの？　区別つくかなぁ。。。
大丈夫！違う服を着ています。

```
void calc1(long double a) {
    printf("long double %f¥n", a);
}
void calc1(double a) {
    printf("double      %f¥n", a);
}
int main()
    calc1(0.1L);          ⇒ long double 0.100000
    calc1(0.1);           ⇒ double      0.100000
}
```

どこで遊ぶ？

x86系のCPUでは、
FPUという演算装置で浮動小数点の演算を行う

FPUには、拡張倍精度（80bit）のレジスタが8本ある
（レジスタ表示で浮動小数点を選択：ST0～ST7）
floatもdoubleも、このレジスタで処理する



かけっこしよっ！

```
C = a + b;
```

【 float 】

```
    fld  dword ptr a        ;aをST0にpush
    fadd dword ptr b        ; ST0にbを加算
    fstp dword ptr c        ; ST0をcに設定してpop
```

【double】

```
    fld  qword ptr a        ;aをST0にpush
    fadd qword ptr b        ; ST0にbを加算
    fstp qword ptr c        ; ST0をcに設定してpop
```

こんなコードならfloatもdoubleも同じ。

かけっこしよっ！

```
float f = 0;
for (int i = 0; i < 100000000; ++i) {
    f += 0.1f;
}
```

```
double d = 0;
for (int i = 0; i < 100000000; ++i) {
    d += 0.1;
}
```

最適化レベル：/O2

float：1056ms　double：233ms

かけっこしよっ！

doubleは・・・

```
    fldz                                                ;st0に0をpush
    fld     QWORD PTR __real@3fb999999999999a  ;st0に0.1をpush
    mov   eax, 10000000
$LN6@main:
    sub    eax, 1
    fadd   ST(1), ST(0)                                 ;st1にst0を足す
    fadd   ST(1), ST(0)
      :
    fadd   ST(1), ST(0)
    fadd   ST(1), ST(0)
    jne     SHORT $LN6@main
```

doubleは、レジスタのみで処理。

かけっこしよっ！

floatは・・・

```
    fldz                                        ;st0に0をpush
    mov   eax, 10000000
    fstp  DWORD PTR [esp]                       ;st0をメモリ(f)に設定してpop
    fld   QWORD PTR __real@3fb99999a0000000     ;st0に0.1をpush
$LN3@main:
    fld   DWORD PTR [esp]                       ;st0にfをpush
    fadd  ST(0), ST(1)                          ;st0にst1を足す
    fstp  DWORD PTR [esp]                       ;st0をfに設定してpop
      :                                         ;floatの精度に変換
    fld   DWORD PTR [esp]
    fadd  ST(0), ST(1)
    fstp  DWORD PTR [esp]
    jne   SHORT $LN3@main
    fstp  ST(0)                                 ;st0をpop
```

かけっこしよっ！

floatは、毎回、計算結果をfloatの精度に変換する。

浮動小数点のコンパイルオプション

/fp:precise・・・デフォルト

/fp:fast

/fp:fastにすれば、処理も速くなり、
精度も良くなり、
プログラムサイズも小さくなる

計算結果の一貫性を保つため。

かけっこしよっ！

```
float f = 0;
for (int i = 0; i < 1000; ++i) {
    f += 0.1f;
}
```

```
float f = 0;
for (int i = 0; i < 1000; ++i) {
    f += 0.1f;
    printf("%.7f", f);
}
```

	printfなし	printfあり
/fp:precise	99.9990463	99.9990463
/fp:fast	100.0000015	99.9990463

VC++ 6.0　　/O2では/fp:fast相当
VC++ .Net 2003　/Op（浮動小数点の整合性を改善する）
VC++ 2005　　/fp:precise

かけっこしよっ！

doubleは80ビットのレジスタで計算しちゃって良いの？

	全体	仮数部
float	32ビット	23ビット
double	64ビット	52ビット
レジスタ	80ビット	64ビット

FPUの演算精度　53ビット

⇒ doubleの仮数部と同じ精度なので変換不要

```
unsigned int control_word;
_controlfp_s(&control_word, _PC_64, _MCW_PC);
_controlfp_s(&control_word, _PC_53, _MCW_PC);
_controlfp_s(&control_word, _PC_24, _MCW_PC);
```

限界超えて遊ぶぞっ！朝までオール？

オーバーフローさせてみよっ！

```
double dd = DBL_MAX;
dd *= 2.0;
dd /= 2.0;

float ff = FLT_MAX;
ff *= 2.0f;
ff /= 2.0f;
```

warning C4756: 定数演算でオーバーフローを起こしました。

限界超えて遊ぶぞっ！朝までオール？

オーバーフローさせてみよっ！

```
float ff = FLT_MAX;
for (int i = 0; i < 12; ++i) {
    ff *= 2.0f;
}
for (int i = 0; i < 12; ++i) {
    ff /= 2.0f;
}
printf("ff = %e¥n", ff);
```

```
double dd = DBL_MAX;
for (int i = 0; i < 12; ++i) {
    dd *= 2.0;
}
for (int i = 0; i < 12; ++i) {
    dd /= 2.0;
}
printf(“dd = %e¥n”, dd);
```

FPUの演算精度：53ビット　/fp:precise

ff = 1.#INF00e+000　　dd = 1.797693e+308

限界超えて遊ぶぞっ！朝までオール？

オーバーフローさせてみよっ！

```
    fld     DWORD PTR __real@7f7fffff                ;st0にFLT_MAXをpush
    fstp    DWORD PTR [esp]                          ;st0をメモリに設定してpop
    fld     QWORD PTR __real@4000000000000000        ;st0に2.0をpush
    mov     eax, 2
$LN6@main:
    sub     eax, 1
    fld     DWORD PTR [esp]                          ;st0にメモリのFLT_MAXをpush
    fmul    ST(0), ST(1)                             ;st0にst1を掛ける（FLT_MAX×2.0）
    fstp    DWORD PTR [esp]                          ;st0をメモリに設定してpop
      :                                              ;オーバーフロー！
```

限界超えて遊ぶぞっ！朝までオール？

オーバーフローさせてみよっ！

```
    fld     QWORD PTR __real@7fefffffffffffff          ;st0にDBL_MAXをpush
    fld     QWORD PTR __real@4000000000000000          ;st0に2.0をpush
    mov     eax, 2
$LN12@main:
    sub     eax, 1
    fmul    ST(1), ST(0)                               ;st1にst0を掛ける（DBL_MAX×2.0）
      :
    jne     SHORT $LN12@main
    fstp    ST(0)                                      ;st0をpop

    mov     eax, 2
    fld     QWORD PTR __real@3fe0000000000000          ;st0に0.5をpush
$LN9@main:
    sub     eax, 1
    fmul    ST(1), ST(0)                               ;st1にst0を掛ける
      :
    jne     SHORT $LN9@main
```

限界超えて遊ぶぞっ！朝までオール？

オーバーフローさせてみよっ！

```
double dd = DBL_MAX;
for (int i = 0; i < 12; ++i) {
    dd *= 2.0;
}

for (int i = 0; i < 12; ++i) {
    dd /= 2.0;
}
printf("dd = %e¥n", dd);
```

dd = 1.797693e+308

```
double dd = DBL_MAX;
for (int i = 0; i < 12; ++i) {
    dd *= 2.0;
}
printf("dd = %e¥n", dd);
for (int i = 0; i < 12; ++i) {
    dd /= 2.0;
}
printf("dd = %e¥n", dd);
```

dd = 1.#INF00e+000

/fp:precise ⇒ /fp:strict

dd = 1.#INF00e+000

dd = 1.#INF00e+000

でもオールは疲れるよ。。。

```
double d = 0;
for (int i = 0; i < 100000000; ++i) {
    d += 0.1;
}
```

```
double d = HUGE_VAL;
for (int i = 0; i < 100000000; ++i) {
    d += 0.1;
}
```

0から加算：233ms　　HUGE_VALから加算： 37742ms

でもオールは疲れるよ。。。

オーバーフローや0割りでは、例外は発生しない。

例外を発生させるには・・・

```
_controlfp_s(&control_word, _MCW_EM & ~_EM_OVERFLOW, _MCW_EM);
_controlfp_s(&control_word, _MCW_EM & ~_EM_ZERODIVIDE, _MCW_EM);
```

try/catchでは捕まえれない。

⇒ ・ __try/__exceptを使う

・/EHsc（C++の標準の例外あり）を

/EHa（構造化例外SEHあり）に変更する

違う公園も行こー！

SSE：Pentium III　SSE2：Pentium4

128bitのレジスタ8本を追加

浮動小数点演算のSIMD処理を行う

SSE ：1レジスタに4個の単精度データを格納・演算

SSE2: 1レジスタに2個の倍精度データを格納・演算

レジスタ表示でSSEを選択 ：XMM0～XMM7

XMM0：XMM00～XMM03

レジスタ表示でSSE2を選択：XMM0～XMM7

XMM0：XMM0DL,XMM0DH

x64の浮動小数点演算はこっち

x86でも/arch:SSE /arch:SSE2で使用できる

違う公園も行こー！

/fp:preciseではオーバーフローしなかったコード

```
double dd = DBL_MAX;
for (int i = 0; i < 12; ++i) {
    dd *= 2.0;
}
for (int i = 0; i < 12; ++i) {
    dd /= 2.0;
}
printf(“dd = %e¥n”, dd);
```

/fp:precise　/arch:SSE2でコンパイルすると・・・

dd = 1.#INF00e+000

違う公園も行こー！

```
    movsd           xmm1, QWORD PTR __real@7feffffffffffff
    movsd           xmm0, QWORD PTR __real@4000000000000000
    add   esp, 20
    mov   eax, 2
    npad 3
$LL12@main:
    sub   eax, 1
    mulsd           xmm1, xmm0
        :
    mulsd           xmm1, xmm0
    jne   SHORT $LL12@main
        :
```

違う公園も行こー！

/fp:preciseではめっちゃ遅かったコード

```
double d = 0;
for (int i = 0; i < 100000000; ++i) {
    d += 0.1;
}
```

```
double d = HUGE_VAL;
for (int i = 0; i < 100000000; ++i) {
    d += 0.1;
}
```

/fp:precise　/arch:SSE2でコンパイルすると・・・

0から加算：193ms　　HUGE_VALから加算：192ms

さっきのかけっこでズルしたっしょ？

```
float f = 0;
for (int i = 0; i < 100000000; ++i) {
    f += 0.1f;
}
```

```
double d = 0;
for (int i = 0; i < 100000000; ++i) {
    d += 0.1;
}
```

f = 2097152.0　　d = 9999999.9811294507

さっきのかけっこでズルしたっしょ？

```
float ff = 0;
for (int i = 0; i < 10000; ++i) {
    f = 0;
    for (int j = 0; j < 10000; ++j) {
        f += 0.1f;
    }
    ff += f;
}
```

f f= 9999754.0

さっきのかけっこでズルしたっしょ？

20971521回目のループ

$LN3@main:

ST0 = +1.0000000149011611e-0001

```
fld     DWORD PTR [esp]     ;st0にfをpush
```

ST0 = +2.09715200000000000e+0006　ST1 = +1.0000000149011611e-0001

```
fadd    ST(0), ST(1)        ;st0にst1を足す
```

ST0 = +2.09715210000000014e+0006　ST1 = +1.0000000149011611e-0001

```
fstp    DWORD PTR [esp]     ;st0をfに設定してpop
```

ST0 = +1.0000000149011611e-0001

```
fld     DWORD PTR [esp]     ;st0にfをpush
```

ST0 = +2.09715200000000000e+0006　ST1 = +1.0000000149011611e-0001

さっきのかけっこでズルしたっしょ？

浮動小数点数値を加算するときに発生する誤差：情報落ち

```
double d = 0;
for (int i = 0; i < 100000000; ++i) {
    d += 0.1;
}
d += 1.0E16;
```

```
double d = 1.0E16;
for (int i = 0; i < 100000000; ++i) {
    d += 0.1;
}
```

後から加算：　d = 1.0000000010000000e+016

初期値に設定：　d = 1.0000000000000000e+016

さっきのかけっこでズルしたっしょ？

```
    fld     QWORD PTR __real@4341c37937e08000
```

ST0 = +1.0000000000000000e+0016

```
    fld     QWORD PTR __real@3fb999999999999a
```

ST0 = +1.0000000000000000e-0001　ST1 = +1.0000000000000000e+0016

```
    add     esp, 12
    mov     eax, 10000000
$LN3@main:
    sub     eax, 1
    fadd    ST(1), ST(0)
```

ST0 = +1.0000000000000000e-0001　ST1 = +1.0000000000000000e+0016

```
    fadd    ST(1), ST(0)
    fadd    ST(1), ST(0)
      :
```

遊び足りない？

浮動小数点数値を減算するときに発生する誤差：桁落ち

```
double d1 = 0.1234567;
double d2 = 0.1234566;
double dd = d1 - d2;
```

```
d1 = 1.234567016363144e-001
d2 = 1.234565973281860e-001
dd = 1.043081283569336e-007　・・・有効桁数が小さくなる
```