

2021 計算機プログラミング演習

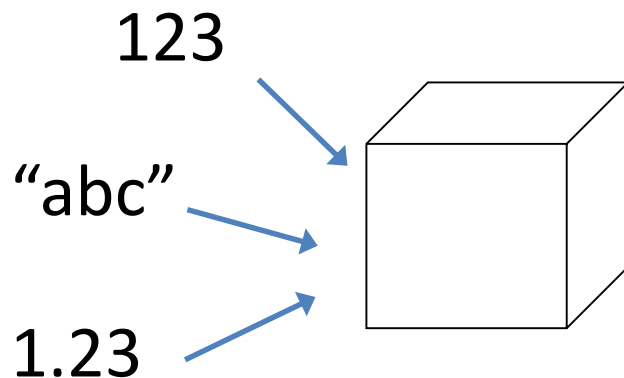
第7回
配列

第7回の目標

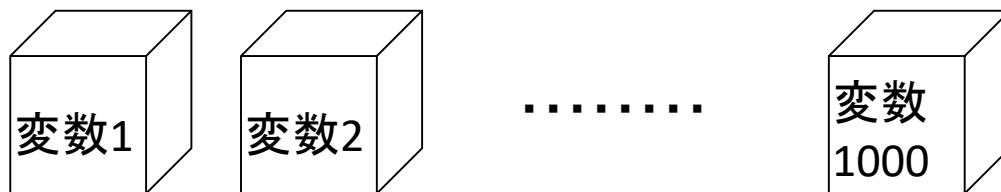
- 配列
 - 概略
 - 変数に順番をつける
 - `for` や `while` などの繰り返しに適したデータ構造
 - 宣言の方法
 - 配列の個数は定数で宣言する
 - 使用に関する注意
 - 配列を使う前に初期化が必要な場合がある。
 - 添字 (`index`) の範囲

配列とは (1/3)

本講義では、変数は決められたデータ型のデータを入れることができる箱のようなものと教えた



では、次のような計算を行うことを考えよう. ランダムな整数の値を持つ1000個の整数の和を求めたい.

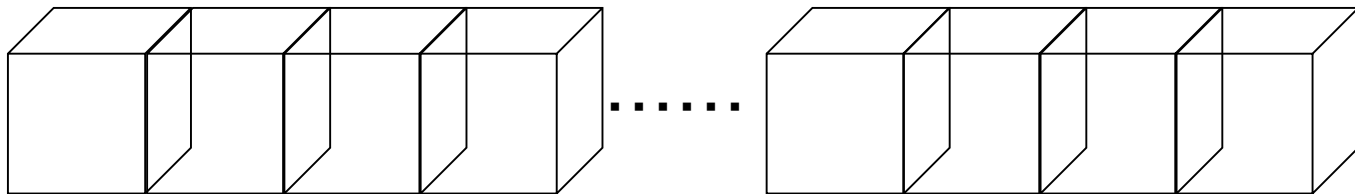


この場合、変数宣言を1000回行い、1000個の変数を用意しなければならない？

配列とは (2/3)

変数の宣言を1000回を行うのは、記述が非常に大変であるだけでなく、プログラムも無駄に長くなってしまい非常に見づらい

以下のように同じデータ型を持つ多くの変数を、一度だけの宣言で扱えないだろうか？



実は、C言語を含む多くのプログラミング言語には、上記のような要望を満たす“配列”と呼ばれるものが用意されている

配列とは (3/3)



配列の宣言

型名 配列名 [要素数]:

例) int seisu[1000]:

double syosu[100]:

基本的には、変数の宣言と同じだが、[]の中の要素数が配列の数である

注意) 配列の番号(添え字)はゼロから始まる数字である
したがって、要素数が1000の場合、添字は0~999である

配列 (1)

- これまで変数は変数名のみによって区別されてきた。
 - 変数 a、変数 b、変数 c とあった場合、「2番目の変数」のような指定の方法はできない。
- 同じような意味の変数を複数用意し、順番(番号)をつけて区別する。
 - 人が10名いて、0番から9番を割り振る。
 - shinchou[10] という配列を用意して、各人の身長データを代入する。
 - k 番の人の身長は shinchou[k] で参照できる。
 - データ処理が簡単になる。

配列の宣言

```
/* dimension.c */
#include <stdio.h>

int main( void )
{
    int          a[ 10 ] ;
    double       x[ 20 ] ;

    return 0;
}
```

整数の配列 `a` と倍精度実数の配列 `x` を作成している。
配列で扱うデータの数はいずれも10個と20個である。
[] で囲まれる部分を添字 (index) と言い、使用できる範囲
は `a[0] ~ a[9]`、`x[0] ~ x[19]`

配列(2)

変数名で区別したプログラム

```
/*      shinchol.c      */
#include <stdio.h>
int main(void) {
    double student1, student2, student3, student4, student5 ;
    double ave;

    student1 = 185 ;
    student2 = 182 ;
    student3 = 181 ;
    student4 = 186 ;
    student5 = 186 ;

    ave = (student1 + student2 + student3 + student4 +
           student5 ) / 5 ;

    printf( "平均身長  %f¥n", ave ) ;
    return 0 ;
}
```

変数をたくさん宣言しなければならぬ

配列(3)

背番号で区別したプログラム(1)

```
/*      shincho2.c      */
#include <stdio.h>
int main(void) {

    double shincho[5];
    double sum = 0 ;
    int i = 0;

    shincho[ 0 ] = 185 ;
    shincho[ 1 ] = 182 ;
    shincho[ 2 ] = 181 ;
    shincho[ 3 ] = 186 ;
    shincho[ 4 ] = 186 ;

    for ( i = 0 ; i <= 4 ; i++ ){
        sum = sum + shincho[ i ] ;
    }
    printf( "平均身長  %f¥n", sum / 5 ) ;
    return 0 ;
}
```

要素数は5 ただし添え字
は0～4であることに注意

配列(4)

背番号で区別したプログラム(2)

```
/*      shincho3.c      */
```

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

```
int main(void) {
```

配列の初期値代入

```
    double shincho[6] = { 0, 185, 182, 181, 186,  
                          186 } ;
```

```
    double sum = 0 ;  
    int i = 0;
```

要素数は6 ただし添え字は0～5である

```
    for ( i = 1 ; i <= 5 ; i++ ) {  
        sum = sum + shincho[ i ] ;  
    }
```

添え字1～5の配列
に存在する身長
データで平均身長
を算出

```
    printf( "平均身長  %f¥n", sum / 5 ) ;  
    return 0 ;
```

```
}
```

配列(5)

背番号で区別したプログラム(3)

```
/*      shincho4.c      */

#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
int main(void) {

    double shincho[6];
    double sum = 0 ;

    for ( i = 1 ; i <= 5 ; i++ ) {
        printf("背番号 %d の身長：" ) ;
        scanf("%lf", &shincho[ i ] ) ;
        sum = sum + shincho[ i ] ;
    }

    printf( "平均身長  %f¥n", sum / 5 ) ;
    return 0 ;
}
```

話は配列から少し外れますが
定義したばかりの変数の中身はどうなっている？

```
/* dim_test2.c */
#include <stdio.h>

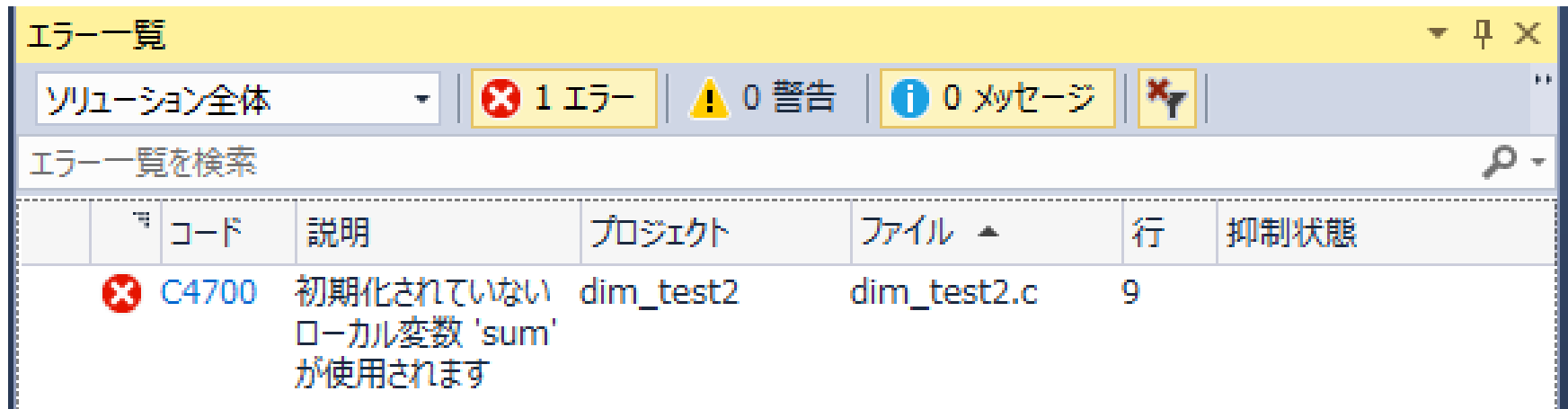
int main( void )
{
    int i, sum;

    printf("sum = %d\\n", sum) ;

    sum = 0;
    for (i = 1; i <= 10; i++) {
        sum = sum + i;
    }

    printf("SUM = %d\\n", sum);
    return 0;
}
```

Visual Studio では ビルド時にエラー と教えてくれる。(親切にも)



error C4700:

初期化されていないローカル変数 'sum' が使用されます

変数が宣言 (`int sum ;`) されてはいるが、
「値を代入していないので、意味のない数値が
使われることになりますよ」 と言っている。

変数の初期化

```
/* dim_test2.c */

#include <stdio.h>

int main( void )
{
    int i, sum = 0 ;

    printf("sum = %d¥n", sum);

    for (i = 1; i <= 10; i++) {
        sum = sum + i;
    }

    printf("SUM = %d¥n", sum);
    return 0;
}
```

では配列の場合は？

```
/* dim_test3.c */
```

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

```
int main( void )
```

```
{
```

```
    int    i;
```

```
    int    x[ 10 ] ;
```

```
    for ( i = 0; i <= 9; i++) {  
        printf( "%d : %d\n", i, x[ i ] );  
    }
```

```
    return 0;
```

```
}
```

ビルド成功！ 実行は？...

1>----- ビルド開始: プロジェクト:dim_test3, 構成:Debug Win32 -----

1> dim_test3.c

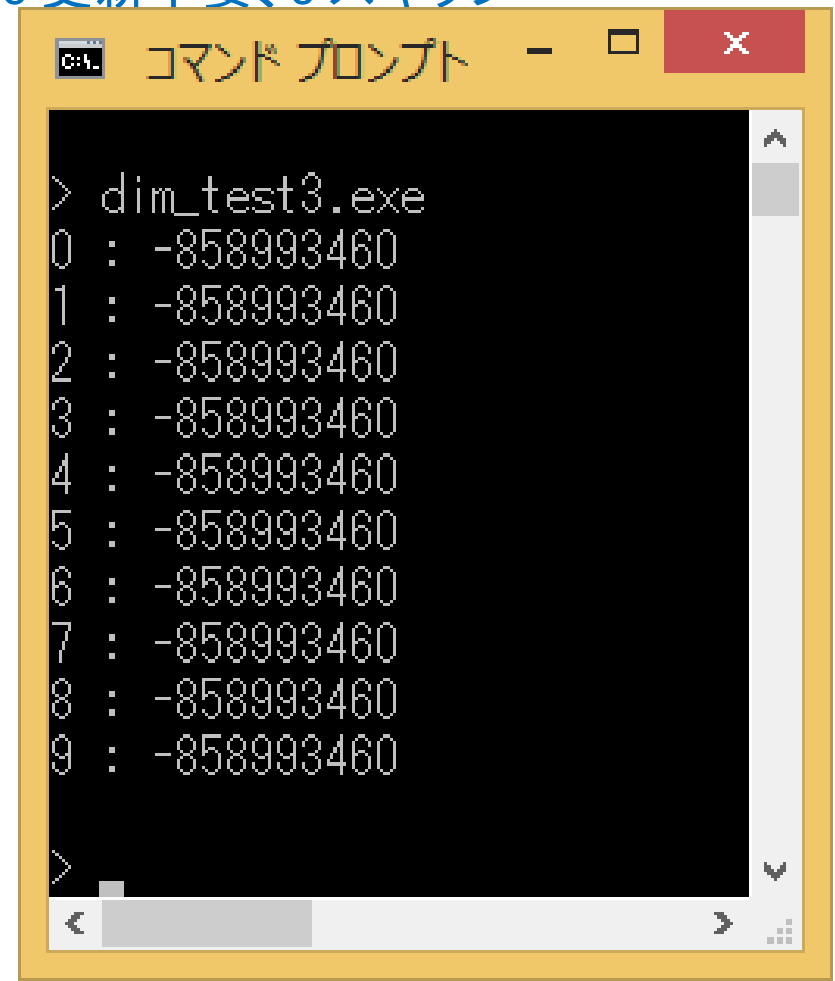
1> dim_test3.vcxproj -> C:¥ ... ¥Debug¥dim_test3.exe

1> dim_test3.vcxproj -> C:¥ ... ¥Debug¥dim_test3.pdb (Partial PDB)

===== ビルド: 1 正常終了、0 失敗、0 更新不要、0 スキップ =====

ビルドでは変数の場合と異なり、**エラーは出ない。**

しかし、配列の値は初期化される(例えば0になる)わけではなく、めちゃくちゃな値が入っているので注意。



```
> dim_test3.exe
0 : -858993460
1 : -858993460
2 : -858993460
3 : -858993460
4 : -858993460
5 : -858993460
6 : -858993460
7 : -858993460
8 : -858993460
9 : -858993460
>
```


変数、配列の初期化について

- C言語では変数や配列を宣言しても、メモリに**場所を確保**するだけである
 - 変数や配列を(ユーザが使いやすいように)**ゼロにしてくるわけではない**。
 - Fortran や BASIC などの言語では自動的にゼロクリアしてくれる
 - Cに影響を受けた言語、C++, Java, C# 等は同様にゼロにしない。
 - 変数や配列に値を代入する前にはめっちゃくちゃな値が入っているので注意する。
 - 配列の内容をゼロにするプログラムに慣れる。
 - 「めっちゃくちゃな値」は変数に割り当てられたメモリの内容がそのまま表示されているだけである。

配列の初期化

```
/* dim_test3.c */  
#include <stdio.h>
```

```
int main( void )  
{
```

```
    int    i;  
    int    x[ 10 ] ;
```

必要に応じて、使用する前に
配列の内容を0にしておく

```
for (i = 0; i <= 9; i++) {  
    x[i] = 0;  
}
```

```
for ( i = 0; i <= 9; i++) {  
    printf( "%d : %d¥n", i, x[ i ] );  
}
```

```
return 0;
```

```
}
```

添字が配列の範囲を超える場合

```
/* dim_test3.c */  
#include <stdio.h>
```

```
int main( void )  
{
```

```
    int    i;  
    int    x[10];
```

```
    for (i = 0; i <= 9; i++) {  
        x[i] = i;  
    }
```

```
    for (i = -5; i <= 15; i++) {  
        printf("%d : %d\n", i, x[i]);  
    }
```

```
    return 0;
```

```
}
```

何のエラーも生じない！

1>----- ビルド開始: プロジェクト:dim_test3, 構成:Debug
Win32 -----

1> dim_test3.c

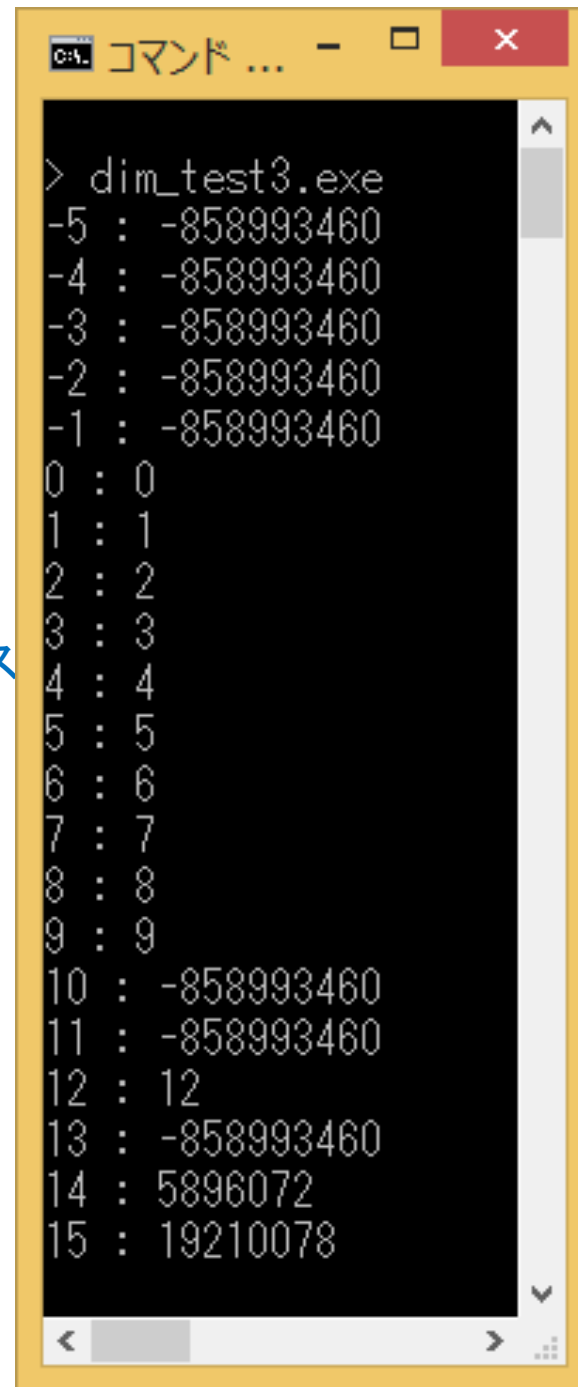
1> dim_test3.vcxproj -> C:¥ ... ¥Debug¥dim_test3.exe

1> dim_test3.vcxproj -> C:¥ ... ¥Debug¥dim_test3.pdb
(Partial PDB)

===== ビルド: 1 正常終了、0 失敗、0 更新不要、0 ス
キップ =====

宣言された範囲の値だけは正しい。

エラーは起きないが、
範囲外の値はめちゃくちゃな値である。



```
コマンド ...  
> dim_test3.exe  
-5 : -858993460  
-4 : -858993460  
-3 : -858993460  
-2 : -858993460  
-1 : -858993460  
0 : 0  
1 : 1  
2 : 2  
3 : 3  
4 : 4  
5 : 5  
6 : 6  
7 : 7  
8 : 8  
9 : 9  
10 : -858993460  
11 : -858993460  
12 : 12  
13 : -858993460  
14 : 5896072  
15 : 19210078
```

「エラーは出ないんだね」 (調子に乗っていると...)

```
/* dim_test3.c */
#include <stdio.h>

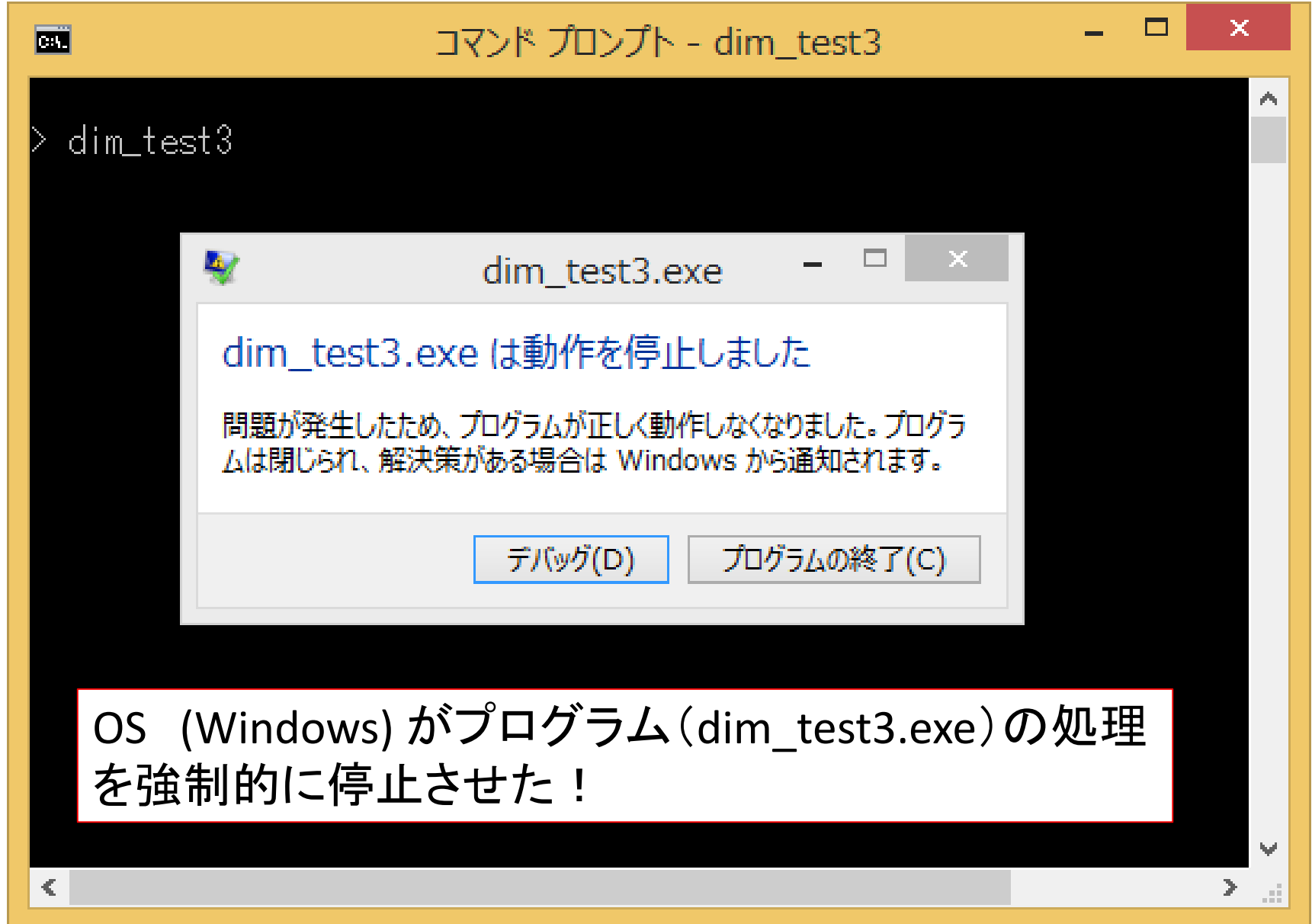
int main( void )
{
    int    i;
    int    x[10];

    for (i = 0; i <= 9; i++) {
        x[i] = i;
    }

    i = 10000;
    printf("%d : %d\n", i, x[i]);

    return 0;
}
```

レッドカード！（退場！！）



なぜ問題が起こったのか？

- 「セグメント例外」
 - マルチタスク OS においては常に複数のプロセス(プログラム)が同一コンピュータ内で実行されている。
 - 個別のプロセスは実行開始時に使用するメモリ範囲(セグメント)が割り当てられる。
 - 割り当てられた範囲外のメモリに値を代入したり、参照したりした場合、他プロセスの実行に問題が生じる。
 - そのため、範囲外のメモリへのアクセスが生じた場合、OS は該当するプロセスを「強制終了」させる。

配列を使う上での重要なポイント

- 配列定義の方法と実際に使える添え字の範囲を理解する。
- 配列を宣言したばかりの状態では配列の内容は「めちゃくちゃ」。初期化が必要か考える。
- 配列の添え字(インデックス)は定義した範囲を超えてはならない。
 - 添え字(インデックス)の管理はプログラマが責任を持たなければならない。