

ソフト系 C言語実習課題 2

祝日(春分の日、秋分の日も含む)を入れた 万年カレンダー

V3.06

2021 JTEC m.h

1

C言語研修課題の3題目は、万年カレンダーの作成です。

このカレンダーのプログラムは、春分の日、秋分の日の祝日も含めた完璧なカレンダーで、西暦で年を与えると、何年のカレンダーでも作成できるという優れものになります。

九九の表では1次元配列を、ライフゲームでは2次元配列を使いました。カレンダーでは、3次元配列を使います。

本課題は、4つのSetpに分けて実習します。

■ 実習を通じて、StepごとにC言語の基礎と下記のことを学びます。

–Step1:

- ✦ 与えられた、年の1月1日の曜日を求める
- ✦ 自分の誕生日(年月日)を入力して、誕生日の曜日を求める
 - 1月～12月までの各月の日数を定義(31,28,31,30,31,30,31,31,30,31,30,31)
 - うるう年の判定をして、2月の日数を28日か29日に変更
 - 該当する年の1月から誕生日の前月までの月の日数を足し、最後に誕生日を足す。

–Step2:

- ✦ 表示したい西暦の年だけを入力
- ✦ 3次元配列を定義し、与えられた年の1年間分のカレンダーを作成する。

–Step3:

- ✦ 春分、秋分の日を求める
- ✦ 祝日マークを加える。
- ✦ ハッピーマンデー(Happy Monday)の処理をする。

–Step4:

- ✦ 横2ヶ月、縦6ヶ月、あるいは横3ヶ月、縦4ヶ月のカレンダーを表示する

2021 JTEC m.h

2

カレンダーの作成も、4つのステップで進めていきます。実際のカレンダーの作成は、Step2からになります。

Step1:

与えられた、年の1月1日の曜日を求めます。

カレンダーの作成には、直接関係しませんが、カレンダーの計算方法の理解を深めるために、自分の誕生日(年月日)を入力して、下記の手順で誕生日の曜日を求めてみます。

(1) 1月～12月までの各月の日数を定義(31,28,31,30,31,30,31,31,30,31,30,31)

(2) 入力した年の前年の12/31日までの累積日数を求める

(3) うるう年の判定をして、2月の日数を28日か29日に変更

(4) 該当する年の1月から誕生日の前月までの月の日数を足し、最後に誕生日を足す。

Step2:

表示したい西暦の年だけを入力

3次元配列を定義し、与えられた年の1年間分のカレンダーを作成する。

Step3:

春分、秋分の日を求める

祝日マークを加える。

ハッピーマンデー(Happy Monday)の処理をする。

Step4:

横2ヶ月、縦6ヶ月、あるいは横3ヶ月、縦4ヶ月のカレンダーを表示する

Step1 : 指定した西暦の年月日の曜日を求める

■ Step1で学ぶこと

– 与えられた、年の1月1日の曜日を求める

– 自分の誕生日の曜日を求める

◆ 月データのテーブルを作成する

➢ 1月～12月まで大の月、小の月の定義テーブル

➢ 定義例: `char mdays[N_MONTH]={31,28,31,30,31,30,31,31,30,31,30,31};`

◆ うるう年の判定をして、2月の日にちを28日か29日かを確定

– enumの活用

◆ enumは、連続した数字の定義に使用する

◆ 月と曜日はenumで定義して使用すると、プログラムの可読性がよくなる。

◆ 定義例:

➢ `enum M_LIST {JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC, N_MONTH};`

➢ `enum W_LIST {SUN, MON, TUE, WED, THU, FRI, SAT, N_WEEK};`

◆ 実際のカレンダーのプログラムでは、enumで定義した名前を使う

➢ そのため、下記の例ように、プログラムプログラム中に数字はほとんど登場しない

➢ `for(month = JAN ; month <= DEC ; month++)`

2021 JTEC m.h

3

Step1では、暦の計算方法の基本を学びます。まず、与えられた、年の1月1日の曜日を求めます。

元旦の曜日を求める方法を理解できたら、カレンダーの作成とは直接関係はしないのですが、応用として、自分の誕生日の曜日を求めてみます。

誕生日の曜日を求めるためには、

1月～12月まで大の月、小の月の定義テーブルと、うるう年の判定が必要になります。そのため、月データのテーブルは、下記のように定義します。

定義例: `char mdays[N_MONTH]={31,28,31,30,31,30,31,31,30,31,30,31};`

そして、うるう年の判定をして、2月の日にちを28日か29日かにします。

enumの活用:

enumは、定数に連続した数字を割り当てるものです。enumは、「C言語の基礎の基礎」で詳しく説明をしています。C言語を勉強していく中で、enumについては比較的后半に登場します。しかし、ここでは、enumを積極的に使用します。

特にカレンダーは、月や曜日を何度か使います。そのとき、直接数字で書くと可読性が低下します。enumを使うことで、月や曜日の表記に数字を一切しなくて済みます。

`enum M_LIST {JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC, N_MONTH};`

➔ この例では、JAN=0, FEB=1, MAR=3.... DEC=11, N_MONTH=12 と連番が割り当たられます。

`enum W_LIST {SUN, MON, TUE, WED, THU, FRI, SAT, N_WEEK};`

➔ 同様に、この例では、SUN=0, MON=1, SAT=6, N_WEEK=7 と連番が割り当てられます。

実際のカレンダーのプログラムでは、enumで定義した名前を使ってください。

そのため、下記の例ように、プログラム中に数字はほとんど登場しなく可読性がよくなります。

`for(month = JAN ; month <= DEC ; month++)` または、

`for(month = JAN ; month < N_MONTH ; month++)`

カレンダーの基本(元旦の曜日を求める)

■ ユリウス(Julian)暦とグレゴリオ暦

－ 現在、多くの国では、グレゴリオ暦を使用

■ グレゴリオ暦で、曜日を求める

－ 1. 西暦1年の1月1日から、求める年-1年の12/31日までの日数を求める

－ 2. グレゴリオ暦では、下記のように求める。

- (1) 求める年-1に、365日を掛け、累積日数を求める
- (2) 求める年-1を4で割った数を、累積日数に足す → 4で割り切れたらうるう年
- (3) 求める年-1を100で割った数を、累積日数から引く → 100で割り切れたら平年
- (4) 求める年-1を400で割った数を、累積日数に足す → 400で割り切れたらうるう年
- 上記(1)～(4)の処理で、求める年の前年の12/31日までの累積日数が求まる
- その累積日数に、1を加えると、求める年の1/1日までの日数になる

(1)～(4)の処理は、求める年-1を別名の変数に代入したら、括弧もいらない1行の式で書けるよ

－ 3. 求めた累積日数を7で割り余りが曜日(西暦1年1月1日は、月曜日)

- 0=日、1=月、2=火、3=水、4=木、5=金、6=土

西暦1年から計算をする、累積日数が16ビットで足りないため、EXCELでは、1900年を基準年として計算をしている。そのため、EXCELでは、1900年より前の年は正しく計算されない。

2021 JTEC m.h

4

カレンダーの計算方法は、実はとても簡単なんです。

暦には、「ユリウス(Julian)暦」と「グレゴリオ暦」の2つがあります。ユリウス暦は、欧州の教会などで使われているようですが、現在は旧暦となっています。日本にも、旧暦がありますよね。

ということで、現在、多くの国では、グレゴリオ暦を使用しています。

グレゴリオ暦は、西暦1年1月1日を月曜日として、求める年月日までの累積日数を求め、それを7で割った余りが曜日となります。

累積日数の計算は簡単で、以下の手順で求める年の12/31日までの累積日数が求まります。その日数に+1をしたら元旦までの日数となります。

したがって、実際の求める年までの累積日数は、「求める年-1」で計算します。そして、その日数に+1をしたら、求める年の元旦までの日数と7で割って曜日が求まります。

1. 西暦1年の1月1日から、求める年-1年の12/31日までの日数を求める

2. グレゴリオ暦では、下記のように求める。

- (1) 求める年-1に、365日を掛け、累積日数を求める
- (2) 求める年-1を4で割った数を、累積日数に足す ← 4で割り切れたらうるう年
- (3) 求める年-1を100で割った数を、累積日数から引く ← 100で割り切れたら平年
- (4) 求める年-1を400で割った数を、累積日数に足す ← 400で割り切れたらうるう年

上記(1)～(4)の処理で、求める年の前年の12/31日までの累積日数が求まる

その累積日数に、1を加えると、求める年の1/1日までの日数になる

3. 求めた累積日数を7で割り余りが曜日(西暦1年1月1日は、月曜日)

0=日、1=月、2=火、3=水、4=木、5=金、6=土

となります。

月日数と日本の祝日 および 1月1日の曜日確認表

月	月日数	月合計	祝日1	祝日2	祝日3
1月	31	0	1 元旦	第2月曜 (成人の日)	
2月	28	31	11 (建国記念日)	23 (令和天皇誕生日)	
3月	31	59	21日頃 (春分の日)		
4月	30	90	29 (昭和の日)		
5月	31	120	3 (憲法記念日)	4 (緑の日)	5 (子供の日)
6月	30	151	なし		
7月	31	181	第3月曜日 (海の日)		
8月	31	212	11 (山の日)		
9月	30	243	第3月曜日 (敬老の日)	23日頃 (秋分の日)	
10月	31	273	第2月曜 (体育の日)		
11月	30	304	3 (文化の日)	23 (勤労感謝の日)	
12月	31	334	23 (平成天皇誕生日)		

- ・祝日と祝日に挟まれた日は、休日
- ・春分、秋分の日は、別途計算で求める

西暦	数値	曜日	
1995	0	日	1995年1月1日(日)
1996	1	月	1996年1月1日(月)
1997	3	水	1997年1月1日(水)
1998	4	木	1998年1月1日(木)
1999	5	金	1999年1月1日(金)
2000	6	土	2000年1月1日(土)
2001	1	月	2001年1月1日(月)
2002	2	火	2002年1月1日(火)
2003	3	水	2003年1月1日(水)
2004	4	木	2004年1月1日(木)
2005	6	土	2005年1月1日(土)
2006	0	日	2006年1月1日(日)
2007	1	月	2007年1月1日(月)
2008	2	火	2008年1月1日(火)
2009	4	木	2009年1月1日(木)
2010	5	金	2010年1月1日(金)
2011	6	土	2011年1月1日(土)
2012	0	日	2012年1月1日(日)
2013	2	火	2013年1月1日(火)
2014	3	水	2014年1月1日(水)
2015	4	木	2015年1月1日(木)
2016	5	金	2016年1月1日(金)
2017	0	日	2017年1月1日(日)
2018	1	月	2018年1月1日(月)
2019	2	火	2019年1月1日(火)
2020	3	水	2020年1月1日(水)
2021	5	金	2021年1月1日(金)
2022	6	土	2022年1月1日(土)
2023	0	日	2023年1月1日(日)
2024	1	月	2024年1月1日(月)
2025	3	水	2025年1月1日(水)

2021 JTEC m.h

5

元旦の曜日が計算できたら、この右側の表で曜日が正しいかを確認してください。

また、左側の表は、日本の祝日の一覧です。日本は、世界的に見て、祝日の数がとても多い国です。

日本の祝日は、固定日(日にちが固定)の祝日と、月曜日を祝日にするハッピーマンデーとがあります。

ハッピーマンデーとは西暦2000年から実施された祝日で、日にちにあまり関係しない祝日を月曜日するというものです。それまで、日本の祝日は、すべて固定日でした。

ハッピーマンデーの背景には、週休2日が定着し、土曜日に祝日が重なった場合、休みが減ることをなくするためのもので、法律で定められています。

現在、ハッピーマンデーは、成人の日、海の日、敬老の日、体育の日の4日あります。この中で、体育の日以外は、確かに日にちにこだわる必要がない祝日です。体育の日は、1964年に開催された東京オリンピックの開会式の日、10月10日(土)を記念して制定された祝日で固定日に意味がありました。ただ、10月は運動会シーズンで、固定日のこだわりをやめて、運動会がやりやすいようにハッピーマンデーにしたいと考えられます。との当時、オリンピックは秋に開催されていました。

ところで、2016年から8月11日が「山の日」となりました。「海の日」はハッピーマンデーなのに、「山の日」は固定日の祝日です。山の日をハッピーマンデーにしなかった理由として、年によっては、1985年8月12日に起きたJAL123墜落事故と重なってしまうことがあるためようです。

また、日本の祝日に関する法律に、「祝日と祝日に挟まれた平日は、休日にする」というものがあります。この法律によって、5月4日が休日となり、何年かに1回は、9月にシルバーウィークとなる年があります。これは敬老の日が第3月曜となり、水曜日が秋分の日となったとき、この法律が適応されて、火曜日が休日になるためです。

さらに、令和元年は、4月27日～5月6日までの大型連休になりました。これは、令和元年を記念して、国が5月1日を特別な祝日にしたため、この法律が適応され、4月30日と5月1日が休日となりました。

なお、2020年と2021年は、東京オリンピックのため、この祝日一覧で、一部の祝日が別の日に移動になっています。注意をしてください。2020年と2021年の特例です。

誕生日の曜日を求める

■ 求める日が1月1日以降の場合

– 求める月の前月までの累計日数を加える

★ (2月以降は求める年のうるう年判定が必要)。

– その後、求める日を加えて、該当年月日までの累積日数を求める

■ うるう年の判定 (うるう年判定の関数を作成)

– 判定する年が、4で割り切れたらうるう年

– 判定する年が、100で割り切れたら平年

– 判定する年が、400で割り切れたらうるう年

西暦2000年は、400年に一度の
左記のすべての条件が成立した年

2月末日の値を変更する例

```
enum M_LIST{JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC, N_MONTH};
```

```
enum W_LIST{SUN, MON, TUE, WED, THU, FRI, SAT, N_WEEK};
```

```
char mdays[N_MONTH]={31,28,31,30,31,30,31,31,30,31,30,31}; ← グローバル変数にしても良い
```

```
mdays[FEB]=うるう年判定(year) ? 29 : 28;
```

3項演算子(条件演算子)

条件演算子を使うと、右の
if文が1行で表現できる。

```
if (うるう年判定(year) == true )  
    mdays[FEB] = 29;  
else  
    mdays[FEB] = 28
```

2021 JTEC m.h

6

元旦が何曜日かの計算方法が分かったところで、自分の誕生日が何曜日だったかを求めてみましょう。

先の説明で、前年の12/31日までの累積日数の求めました。その日数に、+1をしたら元旦までの累積日数でしたが、+1をせずに、誕生月の前月までの日数と誕生日を足せば、誕生日までの累積日数が求まります。

前月までを足すには、1年間の大の月と小の月のデータが必要です。例えば、下記のように定義します。

```
char mdays[N_MONTH]={31,28,31,30,31,30,31,31,30,31,30,31}; ← グローバル変数でも良い。
```

そして、mdays[JAN]から、「誕生月-1」までを加算すれば良いことになります。

ただし、2月に関しては、その年がうるう年かの判定をして、うるう年であれば、29にする必要があります。

うるう年の判定は関数を作ります。そのとき、判定は求める年です。前年ではないので注意してください。

```
int うるう年の判定(int 判定する年) {  
    int result = 0; // 平年にする  
    求める年が、4で割り切れたらうるう年  
    求める年が、100で割り切れたら平年  
    求める年が、400で割り切れたらうるう年  
    return result;  
}
```

その判定結果で、2月の末日を、29か28にします。

この例では、C言語らしく3項演算子(?記号)を使っていますが、通常のif文でもかまいません。

2月の月末の日にちの変更が終わったら、for文で前月までのデータを加算します。

そして、最後に、誕生日を加算して、7で余りを求めたら自分の誕生日の日曜日となります。

ちなみに、西暦2000年は、単に4年に一度のうるう年だったと思っている人もいますが、実は、400年に一度しかない、すべてのうるう年の条件が成立した年だったわけです。

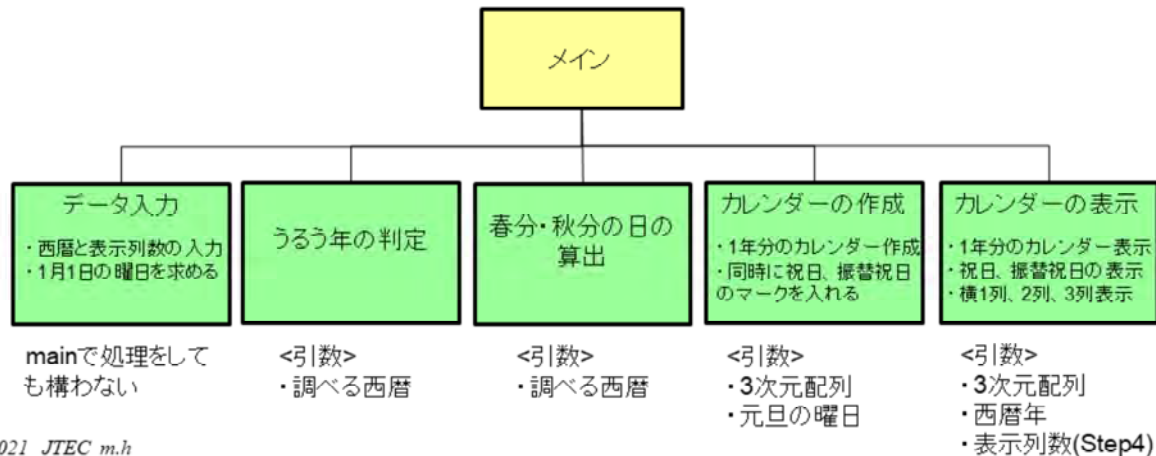
Step2 1年間のカレンダーの作成

元旦の曜日が算出できたところで、実際に1年間分のカレンダーを作成して表示します。
その作成のヒントを解説します。

プログラム構造

■ カレンダーのプログラム構造の例

- データ入力は、メインの中でやってもOK。
- 単にカレンダーの作成であれば、1月1日の曜日が分かれば作成できる
 - 作成する年の1月1日(元旦)を曜日を引数で渡す(年は不要)。
 - メインでループをすれば、何年分のカレンダーも作成ができる
 - うるう年判定、春分・秋分の日算出は、毎年必要なので関数にする



8

プログラムを作成するとき、言語に関係なくすべてに共通することとして、役割ごとにモジュール化をします。**C言語**でのモジュールとは、関数を意味します。課題の要求仕様から、このようなモジュール構造を書くのが良いでしょう。

それぞれのモジュールを**main()**から呼び出します。データ入力は、**main**の中でやってもかまいません。その他は、すべて関数にしてください。

カレンダーの作成は、1月1日(元旦)の曜日が分かれば、曜日は7日周期で繰り返されるので、連続して何年間分も作成できます。

「データ入力」: 表示する年を入力します。これはメインの中でやってもOKですが、**Step4**では、横に何列表示するかの入力も必要になります。

「うるう年の判定」: 引数として調べる西暦年を受け取ります。うるう年の判定は毎年行い、2月の末日を29か28に変更します。

「春分・秋分の日算出」: 引数として調べる西暦年を受け取ります。春分・秋分の日も毎年算出して、祝日テーブルを更新します。

「カレンダーの作成」: 引数として3次元配列と元旦の曜日をを受け取ります。3次元配列と日にちを格納していきます。**Step3**では、祝日データも加えていきます。

「カレンダーの表示」: 引数として3次元配列と西暦の年を受け取ります。日にちが格納されている3次元配列の内容を出力します。**Step3**では、祝日マークも出力します。

このように、モジュール(関数)は、機能ごとに役割が明確になっています。特にカレンダーの作成と表示を分けることで、表示関数を変更だけで複数列表示、カラー表示ができます。表示以下は一切変更はありません。逆に、祝日が変わったりした場合は、祝日テーブルの変更だけで済みます。

1ヶ月分の2次元配列のカレンダーテーブル

6行 x 7列の2次元配列

	日	月	火	水	木	金	土
0					1	2	3
1	4	5	6	7	8	9	10
2	11	12	13	14	15	16	17
3	18	19	20	21	22	23	24
4	25	26	27	28	29	30	31
5							

enumの定義値

SUN	MON	TUE	WED	THU	FRI	SAT
-----	-----	-----	-----	-----	-----	-----

曜日は、配列に不要

赤字の範囲が実際の配列

2021 JTEC m.h

9

これは、1ヶ月分のカレンダーの2次元配列のイメージです。1ヶ月は6行x7列が必要になります。
この図の一番上にある曜日欄は、配列には含まれません。

また曜日は、この図の最下段に示すように、enumで定義したSUN～SATの定数を使用します。

1年分は、この1ヶ月の2次元配列が12枚(1月～12月)まで重なった3次元配列になります。

カレンダーの配列イメージ(3次元配列)

3次元配列の定義例

```
#define WEEKS 6 // 最大行数の定義
char cal_tbl[N_MONTH][WEEKS][N_WEEK];
           ↑      ↑      ↑
           月(12) 行(6) 曜日(7)
```

2021 JTEC m.h

10

この図が、1ヶ月分の2次元配列が12枚重なった3次元配列のイメージになります。3次元配列というと、難しいと思う人がいるかもしれませんが、1次元配列とそんなに変わりません。単に、`[]`の数が増えただけです。EXCELでは、1シートを2次元配列で操作をしています。そのシートが重なれば3次元配列と見なすことができます。

C言語の配列は、`[]`で表します。この`[]`の数が次元になります。

1次元配列では`[]`が1個、2次元配列では`[]`が2個、3次元配列では`[]`が3個になります。

今回のカレンダーの作成で使用する3次元配列は下記のようになります。

```
#define WEEKS 6 // 最大行数の定義 ← enumとは別に定義
char cal_tbl[N_MONTH][WEEKS][N_WEEK];
           ↑      ↑      ↑
           月(12) 行(6) 曜日(7)
```

月(12)と曜日(7)は、enumで定義したN_MONTHとN_WEEKを使います。行のWEEKSは、別途#defineで6を定義してください。

Step2: 1年分のカレンダーを作成

■ Step2で学ぶこと

– 多次元配列の定義

- 1年間分のカレンダーを入れる多次元配列
 - 3次元配列で作成する
- 最初に、3次元配列の中を すべて0 でクリアする ← main関数で初期化をする
- 通常3次元配列の操作は3重ループで行うが、カレンダーの作成は2重ループで行う。
 - 一番外側は、月のループ。その内側は、1日から月末までの日にちのループ
 - 日にちのループの中で、日にちを3次元配列に代入し、その後、曜日を更新
 - 曜日が土曜日を越えたら行を更新する

– 配列の内容を出力(画面に表示)

- 縦12ヶ月で表示する
- 3次元配列の出力は、単純にfor文の3重ループ(すべての要素を出力する)
- そのとき、配列の中が 0 (日付が入っていない)の場合、空白を出力する



黒いコンソール画面の横幅のデフォルトは80文字になっているため、横3ヶ月の表示をすると横幅が超えてしまう。
そこで、コンソール画面の横幅を100文字程度に広げる必要がある。
変更方法は、後述の「付録:コンソール画面の大きさを変更する」を参照のこと。

Step2で、実際に1年間分のカレンダーを作成し表示します。

そのため、カレンダーの作成関数とカレンダーの表示関数の2つを作成します。

まず、main()関数の中で、1年間分のカレンダーを入れる3次元配列を定義します。

そして、定義した3次元配列の中を すべて0 でクリアします。これは、カレンダーの作成で、3次元配列の中に日にちを入れていきますが、日にちは最大で31です、そのため、配列に初期化されない値が残ってしまうためです。

カレンダーの作成関数:

3次元配列の中に、カレンダーの日にちを入れていきます。引数として3次元配列と元旦の曜日を受け取ります。

通常3次元配列の操作は3重ループで行いますが、カレンダーの作成は2重ループで行います。

一番外側は、月のループ。その内側は、1日から月末までの日にちのループになります。

そして、日にちのループの中で、3次元配列の[月][行][曜日]の位置に日にちを代入します。その後、日にちと曜日を更新し、曜日が土曜日を越えたら行を+1して、曜日を日曜日に戻します。

カレンダーの表示関数:

Step2のカレンダーの表示は縦12ヶ月での表示なので簡単です。典型的なfor文の3重ループで、3次元配列の内容を出力(画面に表示)していきます。そのとき、配列の中が 0 (日付が入っていない)の場合、空白を出力します。

そして、曜日のループ(1週間の表示)が終わったら改行します。

カレンダー作成のヒント

■ 2重ループでカレンダーを作成する

ー 一番外側は、1月～12月までのループ

```
for(month=JAN ; month<=DEC ; month++){
```

ここから新しい月が始まる(月初めにすることは、**行を0に、日付を1日**にする)

ー 次のループは、日にちが1日～月末になるまでループ

- 元旦の曜日の位置に、日付(最初は1)を入れる(曜日はmainから引数で受け取る)
- 日にちと曜日をそれぞれ+1する
- 曜日が土曜日を越えたら、行を+1して、曜日を日曜日にする。

	④ 0 日	1 月	2 火	① 3 水	② 4 木	5 金	③ 6 土
0行目				1	2	3	4
⑤ 1行目	5						→
2行目							→
3行目							→
4行目						31	
5行目							

曜日が水曜日(3)の場合、

① 曜日の位置に日付を入れる

② 日付と曜日をそれぞれ+1する

上記、①、②を繰り返す

③ 曜日が土曜を超えたら(曜日>SAT)、

④ 曜日を日曜日戻す(曜日=SUN)

⑤ 行を+1する

上記を月末まで繰り返す

1ヶ月分が終わると、曜日は翌月の1日の曜日に
なっている。

2021 JTEC m.h

12

カレンダーの作成は、mainから、カレンダーの3次元配列と元旦の曜日を引数で受け取ります。
通常3次元配列の操作は3重ループで行いますが、カレンダーの作成は2重ループで行ってみます。
一番外側は、月のループ。その内側は、1日から月末までの日にちのwhileループになります。

```
for(month=JAN ; month<=DEC ; month++){
    // 月初め
    ・ 日にちを1日にする
    ・ 行を0にする
    while ( 日にち <= 該当する月の末日 ){
        :
        :
    }
```

この日にちのwhileループの中で、日にちを3次元配列の [月][行][曜日] の位置に代入します。

日にちを代入したら、日にちと曜日を、それぞれ+1します。

そして、曜日が土曜日(SAT)を超えたら、曜日を日曜日(SUN)に戻し、行を+1します。

この操作で、1年年間分のカレンダーが作成されます。

カレンダー3次元配列出力のヒント

■ 作成したカレンダーの3次元配列の出力

– 単純にfor文の3重ループで配列のすべての要素を出力

```
月のループ{                                     // JAN ~ DEC
    《ここで、年月、曜日表示》
    行のループ{                                     // 0 ~ 6行
        曜日のループ{                               // SUN~SAT
            ここで、日にちが入っていれば日にちを出力、
            日にちが 0 の場合は日にちの代わりに空白を出力
        }
        《1週間分の出力が終わったので、改行する》
    }
}
```

カレンダーの表示は簡単です。mainから、カレンダーの3次元配列と年を引数で受け取ります。通常の3重ループでカレンダーの3次元配列の内容を出力していきます。行のループの前に、年月と曜日の表示をしています。その後、行のループ、曜日をループで3次元配列の内容を出力していきます。このとき、配列の中が 0 の場合は、空白を出力します。

なお、1日あたりの桁は、このあとのStep3で追加する祝日表示を加味して、4桁にしてください。

Step3 祝日を入れる

Step3は、作成したカレンダーに祝日マークを入れ、出力時に祝日マークや振り替え休日のマークを表示するようにします。

この、祝日の処理には、ビット操作で行います。ビット操作を習得してください。

Step3: 祝日を入れる

■ Step3で学ぶこと

ー祝日のデータテーブルを用意する

- 色々な手法がありますが、今回は次ページに示すものを使います
- 計算式によって、春分と秋分の日を求める

ー求めた、祝日をカレンダー配列に入れる(マークする)

- 祝日のマークを、ANDとORのビット操作で行う
 - (1) 最初は、固定日の祝日を入れてみる
 - ・ その日が日曜日なら次の日(月曜日)を振替にする
 - (2) ハッピーマンデーの処理
 - (3) 例外処理(祝日と祝日に挟まれた日は休日)

ー祝日を含めたカレンダー印字

- 祝日は"*"を、振替祝日には"+"を、日付の前に付ける
- 余裕があったら、誕生日マーク(例えば、"&"など)表示する

祝日の処理をするためには、いつが祝日かのデータテーブルを用意します。色々な方法がありますが、今回は、次ページに示すものを使用すると良いと思います。

ただ、春分と秋分の日は、計算で求める必要があり、後述のソースをそのまま使って計算をし、3月の春分の日と9月の秋分の日の日いちを変更します。

祝日の処理とは、カレンダーを作成するときに、ビット操作で祝日マークを入れ、カレンダーの出力時に、祝日マークが入っていれば、日にちを出力する前にマークを表示します。

祝日のマークを入れるときはORで、祝日がマークの有無はANDのビット操作で行います。

祝日の処理をするには、カレンダーの作成関数とカレンダー表示関数の2つを変更をします。

この祝日の処理は、次の手順で、3回に分けて進めていきます。

- (1) 固定日の祝日と日曜日の振り替え休日の処理
- (2) ハッピーマンデーの追加
- (3) 例外処理として、祝日と祝日に挟まれた日は休日にする処理

カレンダーの表示関数では、日にちを表示する前に、祝日マークが入っているかを調べ、該当する祝日マークに応じて、祝日は"*"を、振替祝日には"+"を1文字表示します。祝日マークがない場合は、" "空白を出力します。そのあとに「日にちの下位5ビット」を出力します。 ← 下位5ビットが重要。

Step2では、"%4d"で日にちを表示していました。これは、1日あたり4桁での表示を意味しています。しかし、祝日の処理では、日にちの前に祝日マークが1文字出力されるので、"%3d"として、合計で4桁にする必要があります。ただ、"%3d"だと、祝日マークと日にちとの間が空白が空くので、実際は、"%2d"として、日にちを2桁にしてその、あとに空白を1個入れて4桁にしたほうが見栄えがよくなります。

余裕があったら、誕生日マーク(例えば、"&"など)表示することも良いでしょう。

なお、カレンダーの作成は、3回に分けて祝日を追加していきますが、カレンダーの表示関数は共通ですので、最初の段階で祝日のマークを表示できるようにしておいてください。

祝日のデータテーブル

- 祝日テーブルは、グローバル変数として、main関数の前で定義してもよい。

```
#define MAX_HOLI_TBL 4

//*****
//
// 祝日のデータテーブル
//
char holidays[N_MONTH][MAX_HOLI_TBL] = {
    { 1,-2,0,0},    // 1月、元旦、成人の日
    {11,23,0,0},    // 2月、建国記念日、令和天皇誕生日
    {21, 0,0,0},    // 3月、春分の日(計算で算出)
    {29, 0,0,0},    // 4月、昭和の日
    { 3, 4,5,0},    // 5月、憲法記念日、みどりの日、子供の日
    { 0, 0,0,0},    // 6月、なし
    {-3, 0,0,0},    // 7月、海の日
    {11, 0,0,0},    // 8月、山の日
    {-3,23,0,0},    // 9月、敬老の日、秋分の日(計算で算出)
    {-2, 0,0,0},    // 10月、体育の日
    { 3,23,0,0},    // 11月、文化の日、勤労感謝の日
    { 0, 0,0,0}     // 12月、なし(23日の平成天皇誕生日はなし)
};
```

2021 JTEC m.h

16

これは、祝日データテーブルの例です。

1ヶ月に複数の個の祝日があるので、2次元配列となっています。日本の祝日には、固定日の祝日とハッピーマンデーの祝日があることはすでに説明をしました。そのため、データは固定日の祝日か、ハッピーマンデーかが分かるようにしなければなりません。

そして、データが0の場合は、祝日の処理が終了ということになります。日本では、現在、6月と12月には祝日がありません。すなわち、祝日データが0の場合は、カレンダー作成時に、祝日の処理をスキップできます。

また、この例では、正数を固定日に、負数をハッピーマンデーにしています。固定日の祝日は、数字がそのまま該当する日にちとなります。

一方、ハッピーマンデーは、負数の絶対値が何番目の月曜日であるか意味しています。例えば、1月にある「-2」は、第2月曜日が成人の日であることを意味します。

この例では、ハッピーマンデーを負数にしていますが、100を足して102とか103と言うようにしても構いません。要は、カレンダーの日にちと重ならなければ良いです。30を足して、32とか33でも構いません。

あと、この中で、3月の春分の日と9月の秋分の日は、計算で求めます。

なお、このテーブルを利用するのは、カレンダーを作成するときだけですが、毎年、春分、秋分の日を計算する必要があります。

そのため、グローバル変数として定義してもかまいません。

春分、秋分の日の計算式(下記をコピーして使用)

```
#define SPRING_EQ 0
#define FALL_EQ 1

// 以下を関数にする。引数は、int year
int springEQ;      // 求める春分の日
int fallEQ;        // 求める秋分の日

if (year <= 1899) {
    springEQ = (int) (19.8277 + 0.242194 * (year - 1980.0) - ((year - 1983) / 4));
    fallEQ = (int) (22.2588 + 0.242194 * (year - 1980.0) - ((year - 1983) / 4));
} else if (year >= 1900 && year <= 1979) {
    springEQ = (int) (20.8357 + 0.242194 * (year - 1980.0) - ((year - 1983) / 4));
    fallEQ = (int) (23.2588 + 0.242194 * (year - 1980.0) - ((year - 1983) / 4));
} else if (year >= 1980 && year <= 2099) {
    springEQ = (int) (20.8431 + 0.242194 * (year - 1980.0) - ((year - 1980) / 4));
    fallEQ = (int) (23.2488 + 0.242194 * (year - 1980.0) - ((year - 1980) / 4));
} else if (year >= 2100) {
    springEQ = (int) (21.851 + 0.242194 * (year - 1980.0) - ((year - 1980) / 4));
    fallEQ = (int) (24.2488 + 0.242194 * (year - 1980.0) - ((year - 1980) / 4));
}
holidays[MAR][SPRING_EQ] = springEQ;      // 祝日テーブルに春分の日を入れる
holidays[SEP][FALL_EQ] = fallEQ;          // 祝日テーブルに秋分の日を入れる
```

2021 JTEC m.h

17

このソースは、春分の日と秋分の日を求める計算式です。ちょっと複雑な式ですが、春分の日と秋分の日、天文学に基づいて算出されています。

そのため、内容は理解する必要はありません。

ブラックボックスとして、そのままコピーをして、使ってください。

春分の日と秋分の日を求める関数を作成し、その中にコピーをして入れてください。

引数は、**int year** のみです。

なお、この祝日のデータテーブルには、春分の日を**21日**、秋分の日を**23日**という仮のデータが入れています。そのため、春分の日と秋分の日の算出は、祝日の処理がある程度できてから追加をしても構いません。

また、春分の日と秋分の日の算出は毎年必要になります。同様に、うるう年の判定して2月の末日の変更する処理も毎年必要です。そのため、この**2つ**は、同じタイミングで実施したほうがソースコードの可読性がよくなります。

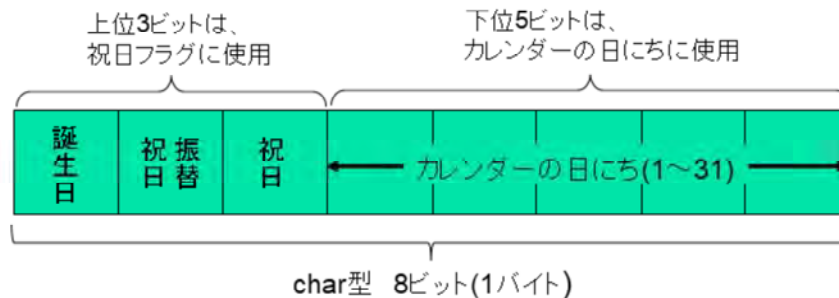
祝日処理のヒント(ビット操作でマークを入れる)

■ カレンダーの日いち(1~31)に必要なビット数は? → 5ビット

– char 型は何ビット? 8ビット(1バイト) → カレンダーのテーブルはchar型で十分

● 配列のサイズは、 $12 \times 6 \times 7 = 504$ バイト。 int型はバイト? → int型すると2016バイト使う

– そのため、上位3ビットは使用しないため、祝日マークに利用する。



下記のようなビットを定義をして、3次元配列のカレンダーテーブルにマークを入れる

```
#define BIRTH_MARK    0x80    // 誕生日マーク  
#define TRANS_MARK    0x40    // 振替祝日マーク  
#define HOLI_MARK     0x20    // 祝日マーク
```

マークを入れるときはビット操作のORを、マークがあるかの確認はANDを利用する

2021 JTEC m.h

18

これは、祝日処理のヒントとして、ビット操作でマークを入れることについて説明です。

まず、カレンダーの日いち、1~31までです。31という数字を表すのに何ビット必要か分かりますか?

16進数は、2進数を4ビットずつに区切って、0~Fの文字で表します。16進数のFは、10進数の15になります。2進数は1ビット増えるごと表現できる数が2倍になります。4ビットは0~15までの16種類の組み合わせです。これを1ビット増やして5ビットにすると、0~31までの32種類の組み合わせになります。

すなわち、5ビットあれば、日いちの1~31までを表現できます。

char型は8ビットなので、カレンダーに使う3次元配列は、char型で十分です。

そして、8ビットのchar型にしても、日いちのデータで使うのは下位5ビットで、上位の3ビットは使用しません。そこで、この使用していない上位3ビットを祝日マークとして利用するのです。

実際は、下記のように各マークを定義します。

```
#define BIRTH_MARK    0x80    // 誕生日マーク  
#define TRANS_MARK    0x40    // 振替祝日マーク  
#define HOLI_MARK     0x20    // 祝日マーク
```

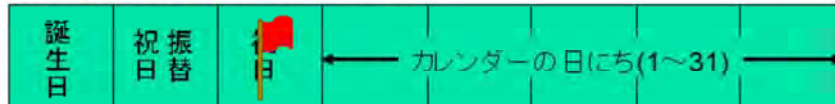
ビット操作

■ ビット操作とは、

- 指定した任意のビット(複数ビットも可)を 0 にしたり 1 にしたりすること → ANDまたはOR操作
- 指定した任意のビット(複数ビットも可)が 0 であるか 1 であるかを調べること → AND操作

■ 祝日マークを入れるとは？

- 祝日マークのフラグをセット(1にする) → OR 操作でビットを1にする
- 変数 |= HORI_MARK; ← ビットをセットする



■ 祝日マークがあるかどうかは？

祝日と定義したビットだけを抽出して、0か1かの判定をする

if(変数 & HORI_MARK) ← ビットがセットされているかを調べる



2021 JTEC m.h

19

ビット操作は、2進数の基本になります。ビット操作とは、

指定した任意のビット(複数ビットも可)を 0 にしたり 1 にしたりすること → ANDまたはOR操作

指定した任意のビット(複数ビットも可)が 0 であるか 1 であるかを調べること → AND操作

祝日マークを入れるとは？

祝日マークのフラグをセット(1にする) → OR 操作でビットを1にすることです。

変数 |= HORI_MARK; ← ビットをセットする

こうすると、図のように指定したビットが1となります(旗が入るイメージ)

実際は、カレンダーの3次元配列に操作をするので、下記のようになります。

カレンダーの3次元配列[月][行][曜日] |= HOLI_MARK; // OR操作でマークを入れる

祝日マークがあるかどうかは？

祝日に定義したビットだけを抽出して、0か1かの判定をします

if(変数 & HORI_MARK) ← ビットがセットされているかを調べる

こうすると、図のように指定したビットのみが抽出されます。

実際は、カレンダーの3次元配列に操作をするので、下記のようになります。

if(カレンダーの3次元配列[月][行][曜日] & HOLI_MARK) // AND操作でマークの有無を検出

注意: C言語では、if文の評価結果が0であればfalse(偽)に、0以外はtrue(真)判定されます。しかし、C++などでは、if文の中は、明示的に論理演算式にしないとエラーになります。論理演算式は、下記のように論理演算子の比較を入れるということです。

if((カレンダーの3次元配列[月][行][曜日] & HOLI_MARK) != 0) ← !=0 が論理演算

祝日マークを入れる実際のプログラム例

```
for (month=JAN ; month<=DEC; month++){
    <月初め>
    日にちを1に、行を0にする ← 今までの処理
    祝日カウンタ = 0;          // 祝日のカウンタをクリア
    月曜日のカウンタ = 0;      // 月曜日のカウンタをクリア

    while (日にち<=月末){
        if (曜日 == MON)      // 今日が月曜日であれば、
            月曜日のカウンタを+1する

        3次元配列[month][行][曜日] |= 日にち;

        if (祝日データ[month][祝日カウンタ] == 日にち){
            3次元配列[month][行][曜日] = HOLI_MARK; // 固定日の祝日が一致なら
                                                    // 祝日マークを入れる
            if (曜日==SUN){
                3次元配列[month][行][曜日+1] = TRANS_MARK; // 今日が日曜日であれば
                                                            // 翌日を振り替えにする
            }

            // ここに、今日が水曜日以上で、祝日と祝日に挟まれた平日は休日にする処理を入れる
            祝日カウンタ++; // 祝日カウンタを進める
        }else{
            // 今日が月曜日であれば、ハッピーマンデーの処理
            祝日カウンタ++;
        }
    }
}
```

祝日が日曜日だった場合、振り替えのマークが先行して入り、日にちはまだ入っていない。そのため、日にちもORで入れる必要がある

2021 JTEC m.h :

20

```
for (month=JAN ; month<=DEC; month++){
    <月初め>
    日にちを1に、行を0にする ← 今までの処理
    祝日カウンタ = 0;          // 祝日のカウンタをクリア
    月曜日のカウンタ = 0;      // 月曜日のカウンタをクリア

    while (日にち<=月末){
        if (曜日 == MON)      // 今日が月曜日であれば、
            月曜日のカウンタを+1する

        3次元配列[month][行][曜日] |= 日にち;

        if (祝日データ[month][祝日カウンタ] == 日にち) // 固定日の祝日が一致なら
            3次元配列[month][行][曜日] = HOLI_MARK; // 祝日マークを入れる
        if (曜日==SUN){
            3次元配列[month][行][曜日+1] = TRANS_MARK; // 今日が日曜日であれば
                                                            // 翌日を振り替えにする
        }

        // ここに、(3)の例外処理(今日が水曜日以上で、祝日と祝日に挟まれた平日は祝日にする処理を入れる)
        祝日カウンタ++; // 祝日カウンタを進める
    }else{
        // 今日が月曜日であればハッピーマンデーの処理
    }
}
```

この例で、3次元配列[month][行][曜日] |= 日にち; と |= で日にちを入れています。これは、祝日が日曜日だったとき、次の日(月曜日)に振り替え休日のマークを入れています。このとき、3次元配列には、まだ日にちが入っていない状態です。この状態で、3次元配列[month][行][曜日] = 日にち; としてしまうと、先行して入れた振り替えマークが消えてしまうため、日にちもORで入れてやります。

すべての祝日の処理が終わったら、2015年を表示してみます。その結果、5月6日に振り替えマークが表示され、9月22日も振り替えて休日となりシルバーウィークになれば正解です。

Step4 カレンダーを複数列で表示する

Step4は、**Step3**で完成したカレンダーを横に2ヶ月ずつ、あるいは3ヶ月と複数列で表示させるというものです。

この処理は、すでに完成しているカレンダーのプログラムで、「カレンダーの表示関数」だけの変更になります。カレンダーの作成などの関数は一切変更しません。

また、余裕があれば、祝日を”*”や”+”といった文字記号の表示でなく、日曜・祝日は赤色、土曜日は青色とカラー表示をすることにも挑戦してみましょう。

カラー表示をする場合も、表示関数だけの変更でできます。その他は関数は変更しません。

Step4: 作成したカレンダーの表示を変える

■ Step4で学ぶこと

- Step3までで、縦12ヶ月で表示していたカレンダーを、横2ヶ月、縦6ヶ月、あるいは横3ヶ月、縦4ヶ月のカレンダーを表示する
- 列の指定は、1列、2列、3列の3種類とする。
 - 横に何列表示するかは、最初に入力して指定する
- 配列のアクセスは、次元数に応じて多重ループになる。
 - 1次元配列は、1重ループ (1次元配列は九九の表で経験済み)
 - 2次元配列は、2重ループ (2次元配列はライフゲームで経験済み)
 - 3次元配列は、3重ループ (3次元配列はカレンダーのStep3で経験済み)
- カレンダーを横に複数列表示するためには、4重ループが必要



ヒント

- 行のループと曜日のループの間に、列のループが入る
- 行のループを -2 から始める方法も考えてみよう
 - 年月、曜日の表示が楽になる

2021 JTEC m.h

22

Step3までで、縦12ヶ月で表示していたカレンダーを、横2ヶ月、縦6ヶ月、あるいは横3ヶ月、縦4ヶ月のカレンダーを表示します。

列の指定は、1列、2列、3列の3種類とします。

年を入力したあとに、横に何列表示するかを入力を行います。

Step3では、単純にfor文の3重ループで、縦1列で12ヶ月表示していました。

- 1次元配列は、1重ループ (1次元配列は九九の表で経験済み)
- 2次元配列は、2重ループ (2次元配列はライフゲームで経験済み)
- 3次元配列は、3重ループ (3次元配列はカレンダーのStep3で経験済み)

カレンダーの複数列表示は、ソースコードの変更は少ないですが、考え方がちょっと複雑になります。カレンダーを横に複数列表示するためには、4重ループとなり、行のループと曜日のループの間に、列のループが入ります。

このとき、行のループは、0から始めていましたが、-2から始める方法も考えてみましょう。

for文の初期を0から始めなくても良いです。

そうすると、年月、曜日の表示の処理が簡単になります。

カレンダーの複数列出力のヒント

■ Step3で表示していた内容を変更する

- 行と曜日のループの間に、列のループを入れる
- 年月と曜日の表示を列ループの中に移動し、改行はしない

```
for 月のループ{           // JAN ~ DEC (月の増加は+1でなく、列数を増加する)
    for 行のループ{         // -2 ~ 6行 (-2から始める)
        for 列のループ{ ← このループを追加(0~指定された列数)
            if (行が-2なら) 年月の表示
            else if(行が-1なら) 曜日の表示 } 年月と曜日の表示は、
            else {           列ループの中です
                for 曜日のループ{ // SUN~SAT
                    日にちが入っていれば日にちを出力、
                    日にちが 0 の場合は日にちの代わりに空白を出力
                }
                ここに月と月の間隔の空白を出力
            }
        }
        列のループが終わったら、改行する
    }
}
```

月の変数に、列ループの変数を足さないと、同じ月のデータが出力される。

これは、実際の例です。青色の部分か° 変更と追加になります。

```
for 月のループ{           // JAN ~ DEC (月の増加は+1でなく、列数分を増加する)
    for 行のループ{         // -2 ~ 6行 (-2から始める)
        for 列のループ{ ← このループを追加(0~指定された列数)
            if (行が-2なら) 年月の表示
            else if(行が-1なら) 曜日の表示
            else {
                for 曜日のループ{ // SUN~SAT
                    日にちが入っていれば日にちを出力、
                    日にちが 0 の場合は日にちの代わりに空白を出力
                }
            }
            列のループが終わったら、改行する
        }
    }
}
```

ここで、ポイントとして、一番最初の月のループは、Step3までは、
`for(month = JAN; month <= DEC; month++)` と月を1ヶ月ずつ増加していましたが、複数列表示をするには、+1でなく、引数で渡された列数を加えます。
そして、月は、列のforループで使う、列変数をプラスします。
つまり、3次元配列の[month + 列の変数][行][曜日]となります。

表示例(横2ヶ月、縦6ヶ月)

2015年 1月 日 月 火 水 木 金 土 4 5 6 7 8 9 10 11 *12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	2015年 2月 日 月 火 水 木 金 土 1 2 3 4 5 6 7 8 9 10 *11 12 13 14 15 16 17 18 19 20 21 22 *23 24 25 26 27 28	2015年 7月 日 月 火 水 木 金 土 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 *20 21 22 23 24 25 26 27 28 29 30 31	2015年 8月 日 月 火 水 木 金 土 2 3 4 5 6 7 8 9 10 *11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
2015年 3月 日 月 火 水 木 金 土 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 *21 22 23 24 25 26 27 28 29 30 31	2015年 4月 日 月 火 水 木 金 土 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 *29 30	2015年 9月 日 月 火 水 木 金 土 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 *21 *22 *23 24 25 26 27 28 29 30	2015年10月 日 月 火 水 木 金 土 4 5 6 7 8 9 10 11 *12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
2015年 5月 日 月 火 水 木 金 土 * 3 * 4 * 5 + 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	2015年 6月 日 月 火 水 木 金 土 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30	2015年11月 日 月 火 水 木 金 土 1 2 * 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 *23 24 25 26 27 28 29 30	2015年12月 日 月 火 水 木 金 土 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

これは、実際の横2列、縦6行の出力例です。

表示例(横3ヶ月、縦4ヶ月)

2015年 1月	2015年 2月	2015年 3月
日 月 火 水 木 金 土	日 月 火 水 木 金 土	日 月 火 水 木 金 土
1 2 3	1 2 3 4 5 6 7	1 2 3 4 5 6 7
4 5 6 7 8 9 10	8 9 10 *11 12 13 14	8 9 10 11 12 13 14
11 *12 13 14 15 16 17	15 16 17 18 19 20 21	15 16 17 18 19 20 *21
18 19 20 21 22 23 24	22 *23 24 25 26 27 28	22 23 24 25 26 27 28
25 26 27 28 29 30 31		29 30 31
2015年 4月	2015年 5月	2015年 6月
日 月 火 水 木 金 土	日 月 火 水 木 金 土	日 月 火 水 木 金 土
1 2 3 4	1 2	1 2 3 4 5 6
5 6 7 8 9 10 11	* 3 * 4 * 5 + 6 7 8 9	7 8 9 10 11 12 13
12 13 14 15 16 17 18	10 11 12 13 14 15 16	14 15 16 17 18 19 20
19 20 21 22 23 24 25	17 18 19 20 21 22 23	21 22 23 24 25 26 27
26 27 28 *29 30	24 25 26 27 28 29 30	28 29 30
	31	
2015年 7月	2015年 8月	2015年 9月
日 月 火 水 木 金 土	日 月 火 水 木 金 土	日 月 火 水 木 金 土
1 2 3 4	1	1 2 3 4 5
5 6 7 8 9 10 11	2 3 4 5 6 7 8	6 7 8 9 10 11 12
12 13 14 15 16 17 18	9 10 *11 12 13 14 15	13 14 15 16 17 18 19
19 *20 21 22 23 24 25	16 17 18 19 20 21 22	20 *21 +22 *23 24 25 26
26 27 28 29 30 31	23 24 25 26 27 28 29	27 28 29 30
	30 31	
2015年10月	2015年11月	2015年12月
日 月 火 水 木 金 土	日 月 火 水 木 金 土	日 月 火 水 木 金 土
1 2 3	1 2 * 3 4 5 6 7	1 2 3 4 5
4 5 6 7 8 9 10	8 9 10 11 12 13 14	6 7 8 9 10 11 12
11 *12 13 14 15 16 17	15 16 17 18 19 20 21	13 14 15 16 17 18 19
18 19 20 21 22 23 24	22 *23 24 25 26 27 28	20 21 22 23 24 25 26
25 26 27 28 29 30 31	29 30	27 28 29 30 31

これは、実際の横3列、縦4行の出力例です。

いずれも、2015年の例です。5月6日が振り替えに、9月22日が休日になっていることを確認してください。

。

Step4.5 おまけ-月と月の間隔を自動化するテクニック

■ 横に表示するとき月と月の空白を自動計算する

－ 月と月の間は、空白を入れて間隔の調整したと推測する

2015年 1月							2015年 2月							2015年 3月						
日	月	火	水	木	金	土	日	月	火	水	木	金	土	日	月	火	水	木	金	土
					*	1	1							1						
4	5	6	7	8	9	10	8	9	10	*11	12	13	14	8	9	10	11	12	13	14
11	*12	13	14	15	16	17	15	16	17	18	19	20	21	15	16	17	18	19	20	*21
18	19	20	21	22	23	24	22	*23	24	25	26	27	28	22	23	24	25	26	27	28
25	26	27	28	29	30	31								29	30	31				

－ その月と月の空白を計算で求める方法

- カレンダーの1日の表示桁を4桁にした場合、1週間に必要な桁数は、
 - 4文字 × 7日 = 28文字
- 年月日の表示は、例えば、"%4d年2d%月"とした場合、年月で使用する桁数は、
 - 漢字は2文字で数えるため、4 + 2(年) + 2 + 2(月) = 10文字
 - すなわち、年月表示の1週間の文字数で足りない空白は
 - 28文字 - 10文字 = 18文字

Step4で、複数列表示を追加するとき、月と月との間の空白など、何度もコンパイルをして位置合わせをしたかと思います。

ここでは、`sprintf()`を使って、その位置合わせを自動的にするテクニックを説明します。

この位置合わせで、手間がかかるのか、年月の表示の行になります。

考えたとして、

カレンダーの1日の表示桁を4桁にした場合、1週間に必要な桁数は、

「4文字 × 7日 = 28文字」になります。

それで、年月日の表示を、

例えば、"%4d年2d%月"とした場合、年月で使用する桁数は、

漢字は2文字で数えるため、4 + 2(年) + 2 + 2(月) = 10文字になります。

すなわち、年月表示の1週間の文字数で足りない空白は、「28文字 - 10文字 = 18文字」になります。

この足りない空白の数を算出するには、表示する年月の文字列の長さかわからなければなりません。そこで、利用するのが、`sprintf()` 関数です。

3つのprintf()

- データの出力に最も多く使用するprintf()は3種類ある

```
printf (    "hello World\n"); → 普通のprintfで画面に出力
fprintf (fp, "hello World\n"); → fpで指定したファイルに出力
sprintf (buf, "hello World\n"); → 出力をbufに格納
```

- 年月日を表示して、月の終わりに満たない空白を求める

- 出力結果をbufに格納

```
char buf[64];
int len;
sprintf(buf, "%4d年%d月", year, month);
```

yearが2021、monthが4であれば、
bufには、"2019年4月"が入る

- 格納した出力の文字数を求める (strlen関数を使う)

```
len = strlen(buf);
```

- 月の終わりまでに足りない空白の数は、 $4*N_WEEK - len$ で求まる

- 指定した数だけスペースを出力する関数を作成する

```
printf("%s", buf); ← bufの内容を文字列として出力する
put_char(' ', 4*N_WEEK - len); ← 足りない空白を出力
```

put_char()関数は、九九の表で横線を表示する時に作成した関数を参考して作成する。

put_char() は、新たに作成する関数です。標準関数のputchr()ではありません。

2021 JTEC m.h

27

今まで、データの出力には、printf()を使ってきました。実は、このprintf()には3種類あります。

```
printf (    "hello World\n");          → 普通のprintfで画面に出力
fprintf (fp, "hello World\n");          → fpで指定したファイルに出力
sprintf (buf, "hello World\n");          → 出力をbufに格納
```

上記の3つのprintf()は、書式指定以降は、すべて同じです。

fprintf() は、第1引数にファイルポインタを指定します。そうすると、実行結果を指定したファイルポインタに出力されます。ファイルポインタには、オープンしたファイルのファイルポインタを指定します。

システムが用意してあるシステム変数(stdout, stderr)も使えます。例えば、

```
fprintf(stderr, "hello World\n");
```

 とすると、結果はエラー出力のファイルポインタに出力されます。ちなみに、普通の printf("hello world"); は、 fprintf(stdout, "hello world"); と同じになります。

sprintf()は、第1引数に文字列のchar型の配列を指定します。そうすると、実行結果を指定したchar型の配列に書き込まれます。例えば、

```
char buf[64];
```

```
sprintf(buf, "%d年%d月", year, month);
```

 とすると、実行結果はbufに格納されます。

yearが2021、monthが4の場合、“2021年4月”という文字列がbufに入ります。そこで、strlen()関数で、buf内の文字列の長さを調べます。strlen()は、ライフゲームでファイルからテキストデータを読込んだとき使いました。

文字列の長さが分かれば、1週間の長さ(28文字)からbuf内の文字列の長さを引くことで、足りない空白数を算出できます。そして、1文字を指定した数だけ表示する関数(この例では、put_char())を作成し呼び出すだけです。

九九の表で、文字列を指定した数だけ表示する関数を作成しましたが、今度は、1文字を指定した数だけ表示する関数になります。

付録:ANSIエスケープシーケンス

ライフゲームでは、エスケープシーケンスで画面のクリアとカーソルの移動を行いました。
カレンダーでは、エスケープシーケンスで祝日のカラー表示をしてみます。

ANSIエスケープシーケンスとは、0x1B(ESC)コードに続く複数の文字列で画面制御を行うものです。
ISOやJISの規格としても定められています。

ANSIとは、American National Standards Institute(米国標準協会)の略です。

エスケープシーケンスを使うことで、文字に色を付けたり、画面のクリア、カーソルの移動などができます。

ライフゲームでは、画面のクリアとカーソルの移動に、エスケープシーケンスを使いましたが、カレンダーでは、祝日のカラー表示をしてみます。

画面の制御 - エスケープシーケンスとは?

- Unixなどの世界では、画面の制御にエスケープシーケンスというものを使用している。
- Windows10から、コマンドプロンプトでエスケープシーケンスがサポートされた。
- エスケープシーケンスとは、0x1bに続く文字で色々と定められている。
- 下記は、その一例です。

```
printf("\033[2J"); //画面クリア
printf("\033[0K"); //カーソル位置からその行の右端までをクリア
printf("\033[1K"); //カーソル位置からその行の左端までをクリア
printf("\033[2K"); //カーソル位置の行をクリア

printf("\033[%d;%dH",10,20); //カーソル位置を、高さ10行目、横20文字目に移動
printf("\033[%dC",10); //カーソルを10文字だけ右に移動
printf("\033[%dD",10); //カーソルを10行文字だけ左に移動
printf("\033[%dB",10); //カーソルを10行だけ下に移動
printf("\033[%dA",10); //カーソルを10行だけ上に移動

printf("\033[30m"); //黒の文字に変更
printf("\033[31m"); //赤の文字に変更
printf("\033[32m"); //緑の文字に変更
printf("\033[33m"); //黄色の文字に変更
printf("\033[34m"); //青の文字に変更
printf("\033[35m"); //マゼンダの文字に変更
printf("\033[36m"); //シアンの文字に変更
printf("\033[37m"); //白の文字に変更
printf("\033[39m"); //文字の色を通常の色に戻す
printf("\033[0m"); //リセット(設定を未設定の状態に戻す)
```

Teratermなどでテストをする場合は、ESCキーを押した後に、「[」「2」「J」と押します。以下同じです。

実際は、各色を定義して使う

```
#define COLOR_BLACK      "\033[30m" // 黒色
#define COLOR_RED        "\033[31m" // 赤色
#define COLOR_GREEN      "\033[32m" // 緑色
#define COLOR_YELLOW     "\033[33m" // 黄色
#define COLOR_BLUE       "\033[34m" // 青色
#define COLOR_MAGENTA    "\033[35m" // 紫色
#define COLOR_CYAN       "\033[36m" // 水色
#define COLOR_WHITE      "\033[37m" // 白色
#define COLOR_NORM       "\033[39m" // 通常に戻す
```

¥033 は、C言語における8進数の定数表現 ¥x1b として、16進数としても良い。
例: "\033[30m" は: "\x1b[30m" と同じ

2021 JTEC m.h

29

このリストは、エスケープシーケンスの一部です。実際のエスケープシーケンスについては、別途ネットなどで調べてください。

エスケープシーケンスは、全体を文字列として指定します。最初の文字は、エスケープ文字「0x1B」で、そのあとに「[」の文字が続き、以後複数の文字で構成されます。ただし、文字列にエスケープ文字を直接書くことはできません。C言語のエスケープ文字「¥」記号を使って入れます。

「エスケープ文字」と「エスケープシーケンス」言う言葉が出てきましたが、両者を混同しないでください。文字列の中に、「0x1B」という文字コードを入れるには、16進数で入れる場合と8進数で入れる場合の二通りがあります。

16進数で指定する場合は、「¥x1B」と書きます。◀ ¥記号の後は「x」1文字だけです、C言語の16進数定数指定「0x」にはしないでください。0は不要です。

一方、8進数で指定する場合は、¥033と書きます。C言語では、0から始まる数字は、8進数として評価します。ネット上で、¥033としている例が多いです。033は、16進数の0x1B、10進数の27になります。

実は、国際規格であるエスケープシーケンスは、長い間Windowsではサポートされていませんでした。しかし、Windows10から標準サポートなり仕使用できるようになりました。

もし、エスケープシーケンスを出力して、画面に、「[2;1H」のような文字が表示される場合は、エスケープシーケンスがサポートされていないか、あるいはコマンドプロンプトの設定がレガシーモードになっている可能性があります。

エスケープシーケンスを使ったカラー表示

■ コマンドプロンプトで、エスケープシーケンスを実行した例

```
R:\>type ansi_test.txt
  m  Color  Color  40m  41m  42m  43m  44m  45m  46m  47m
1m  Color  Color  Color  Color  Color  Color  Color  Color  Color
30m  Color  Color  Color  Color  Color  Color  Color  Color  Color
1;30m Color  Color  Color  Color  Color  Color  Color  Color  Color
1;31m Color  Color  Color  Color  Color  Color  Color  Color  Color
1;32m Color  Color  Color  Color  Color  Color  Color  Color  Color
1;33m Color  Color  Color  Color  Color  Color  Color  Color  Color
1;34m Color  Color  Color  Color  Color  Color  Color  Color  Color
1;35m Color  Color  Color  Color  Color  Color  Color  Color  Color
1;36m Color  Color  Color  Color  Color  Color  Color  Color  Color
1;37m Color  Color  Color  Color  Color  Color  Color  Color  Color

Unsupported ANSI escape codes:
ansi code 2 (faint)
ansi code 3 (italic)
ansi code 4 (underscore)
ansi code 5 (slow blink)
ansi code 6 (fast blink)
ansi code 7 (reverse/negative)
ansi code 8 (conceal/hidden)
ansi code 9 (strike-through)
```

2021 JTEC m.h

30

これは、エスケープシーケンスのテストデータを表示した例です。コマンドプロンプトあるいはPowerShellを起動し、研修サーバー「課題2_カレンダー」にある「escTest.txt」をファイルを、typeコマンドを使って表示させるのこのような画面になります。

typeコマンドは、コマンドプロンプトあるいはPowerShellに内蔵されているコマンドで、テキストファイルを画面に表示するものです。

コマンドプロンプトあるいはPowerShell から、type escTest.txt と入力してみてください。

もし、Windows10で、このようなカラー表示にならない場合は、コマンドプロンプトが「レガシーコンソールを使用する」になっている可能性があります。その確認は、本資料の最後に記載しています。

エスケープシーケンスのテストプログラム

テストプログラム

```
#include<stdio.h>

void main(void) {

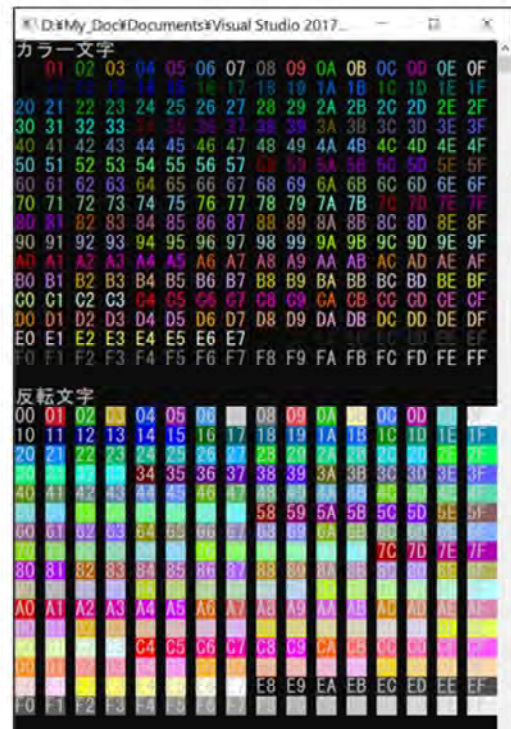
    int row, col, num;

    printf("カラー文字¥n");
    for (row = 0; row < 16; row++) {
        for (col = 0; col < 16; col++){
            num = row * 16 + col;
            printf("¥033[38;5;%dm¥02X¥033[0m", num, num);
        }
        printf("¥n");
    }

    printf("¥n反転文字¥n");
    for (row = 0; row < 16; row++) {
        for (col = 0; col < 16; col++){
            num = row * 16 + col;
            printf("¥033[48;5;%dm¥02X¥033[0m", num, num);
        }
        printf("¥n");
    }
}
```

2021 JTEC m.h

実行結果



31

これは、カラー表示の確認のためのテストプログラムです。このプログラムを実行すると、右側のようなカラー文字が表示されます。

ちなみに、“¥033[0m” は、指定をリセットし未指定状態に戻します。これを忘れると、ずっと色が付いたままになります。

“¥033”は、“¥x1b”としてもOKです。

エスケープシーケンスは、カラー指定だけではなく、ブリンクの設定、カーソルの移動など、とても多くが定義されています。エスケープシーケンスにどんなものがあるかは、ネットで調べてください。

文字による祝日表示

2021年 1月							2021年 2月							2021年 3月						
日	月	火	水	木	金	土	日	月	火	水	木	金	土	日	月	火	水	木	金	土
	3	4	5	6	7	*1	2	7	8	9	10	*11	12	13	7	8	9	10	11	12
10	*11	&12	13	14	15	16	14	15	16	17	18	19	20	14	15	16	17	18	19	*20
17	18	19	20	21	22	23	21	22	*23	24	25	26	27	21	22	23	24	25	26	27
24	25	26	27	28	29	30	28							28	29	30	31			
31																				

2021年 4月							2021年 5月							2021年 6月						
日	月	火	水	木	金	土	日	月	火	水	木	金	土	日	月	火	水	木	金	土
	4	5	6	7	8	9	2	*3	*4	*5	6	7	8	6	7	8	9	10	11	12
11	12	13	14	15	16	17	9	10	11	12	13	14	15	13	14	15	16	17	18	19
18	19	20	21	22	23	24	16	17	18	19	20	21	22	20	21	22	23	24	25	26
25	26	27	28	*29	30		23	24	25	26	27	28	29	27	28	29	30			
							30	31												

2021年 7月							2021年 8月							2021年 9月						
日	月	火	水	木	金	土	日	月	火	水	木	金	土	日	月	火	水	木	金	土
	4	5	6	7	8	9	1	2	3	4	5	6	7	5	6	7	8	9	10	11
11	12	13	14	15	16	17	8	9	10	*11	12	13	14	12	13	14	15	16	17	18
18	*19	20	21	22	23	24	15	16	17	18	19	20	21	19	*20	21	22	*23	24	25
25	26	27	28	29	30	31	22	23	24	25	26	27	28	26	27	28	29	30		
							29	30	31											

2021年10月							2021年11月							2021年12月						
日	月	火	水	木	金	土	日	月	火	水	木	金	土	日	月	火	水	木	金	土
	3	4	5	6	7	8	7	8	9	*10	11	12	13	5	6	7	8	9	10	11
10	*11	12	13	14	15	16	14	15	16	17	18	19	20	12	13	14	15	16	17	18
17	18	19	20	21	22	23	21	22	*23	24	25	26	27	19	20	21	22	23	24	25
24	25	26	27	28	29	30	28	29	30					26	27	28	29	30	31	

2021 JTEC m.h

32

文字による祝日表示例です。

祝日は、"*" で、振り替え休日は、"+"で表示されています。

また、誕生日(1月12日)には、"&"を表示しています。

なお、この例では2015年を表示しています。2015年は、5月6日と9月22日が振り替え休日となっています。作成したカレンダーでこのように表示されるかを確認してください。

カラーによる祝日表示

The screenshot displays a calendar application window titled "D:\My_Doc\Documents\Visual Studio 2017\Projects\CalenColor\Debug\CalenColor.exe". It shows a grid of 12 monthly calendars for the year 2021. Each month's calendar has columns for days of the week (日, 月, 火, 水, 木, 金, 土) and rows for dates. The dates are color-coded: red for holidays and substitute days, blue for weekends, and green for birthdays. For example, in January, the 1st is red, the 12th is green, and the 13th is blue. In May, the 6th is red. In September, the 22nd is red. In December, the 25th is red.

2021 JTEC m.h

33

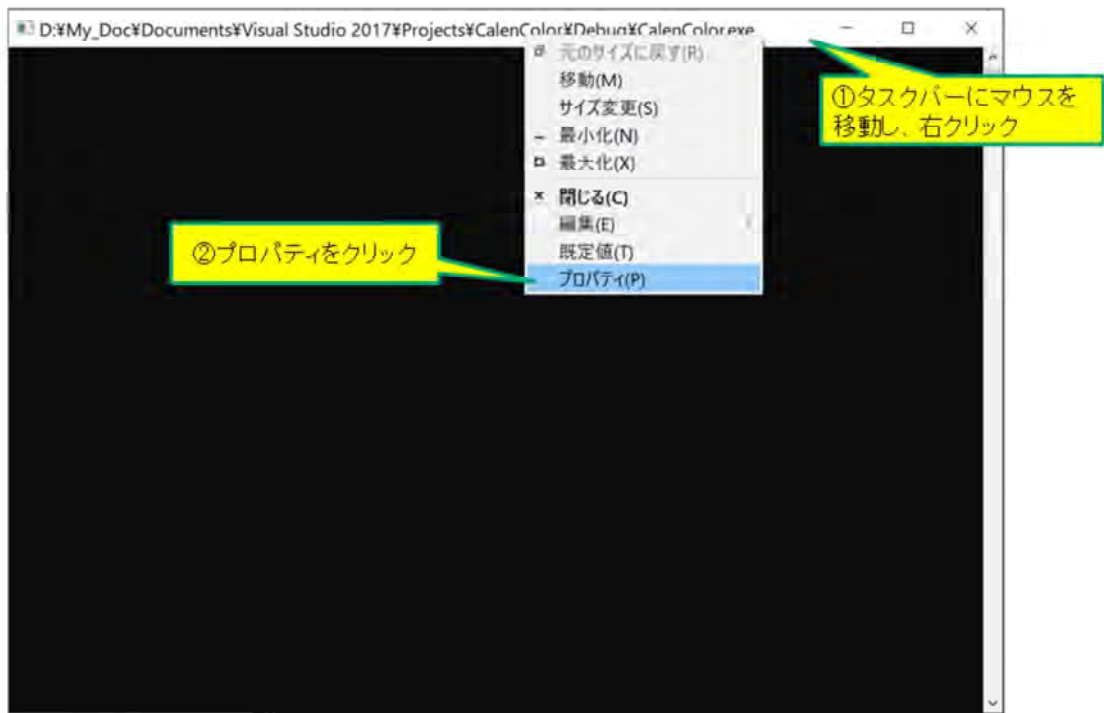
エスケープシーケンスを使ったカラーによる祝日表示例です。

日・祝祭日、振り替え休日は、「赤色」で、土曜日は、「青色」で表示しています。

そして、誕生日(1月12日)は、「緑色」で表示をしています。

文字表示同様に、2015年の5月6日と9月22日が振り替え休日となっているか、自分で作成したカレンダーがこのように表示されるかを確認してください。

カラーが表示されないときの確認 その1

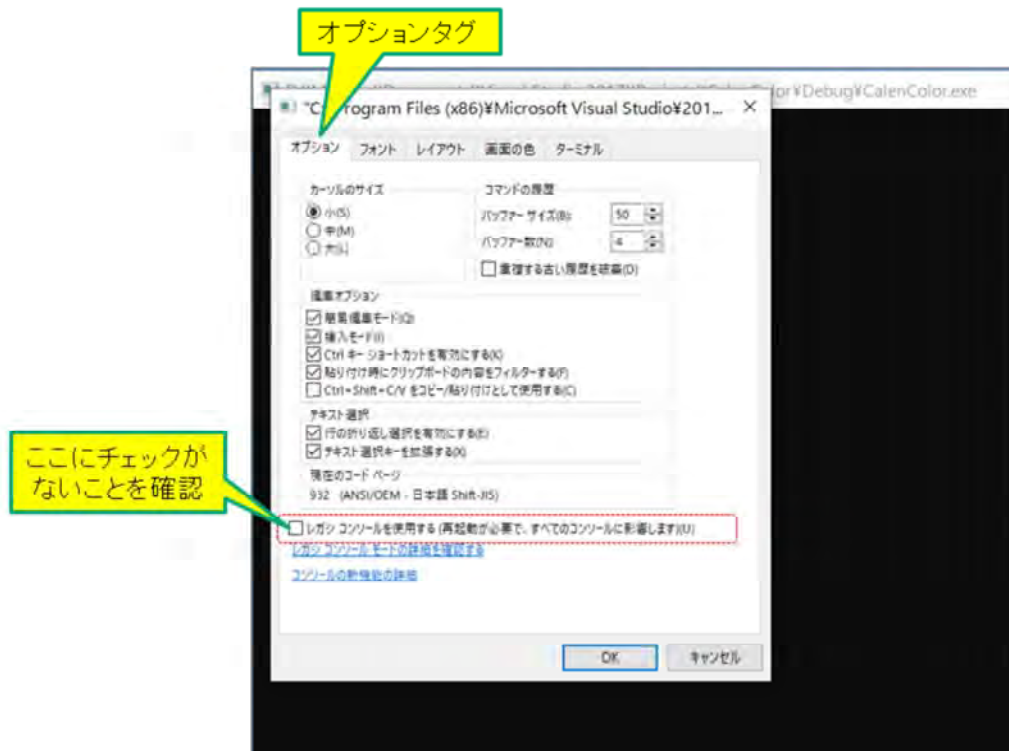


2021 JTEC m.h

34

プログラムの最初行付近にブレークポイントを設定し実行を止めます。黒いコンソールウィンドウが表示されたら、タスクバーで右クリックをします。プルダウンメニューが表示されますので、一番下の[プロパティ]を選択します。

カラーが表示されないときの確認 その2



2021 JTEC m.h

35

プロパティ画面が表示されたら、[オプション]タグを選択し、一番下にある、「☐ レガシーコンソールを使用する(再起動が必要で、すべてのコンソールに影響します)」の先頭に、チェックが入っていないことを確認してください。

チェックが入っている旧バージョンとの互換モードとなり、エスケープシーケンスが動作しません。

もし、チェックが入っているなら、チェックを外し、コンソールの再起動をしてください。